Semantische Modellierung im Kontext Internetfernsehen: Konzeption und Gestaltung eines Prototypen für den Sender Zucker TV

Bachelorarbeit

eingereicht zur Erlangung des akademischen Grades

Bachelor of Science

Erster Gutachter	:	Prof. Dr. Kristian Fischer
Zweiter Gutachter	:	Dipl. Designer Christian Noss

Fachhochschule Köln
Cologne University of Applied Sciences

Fakultät für Informatik und Ingenieurswissenschaften
Studiengang Medieninformatik

Lars Brillert (04.05.1985, Berg. Gladbach)
Stephan Pavlovic (07.10.1984, Olpe)

Gummersbach, August 2007

Zusammenfassung

Das *Semantic Web* ist einer Erweiterung des aktuellen World Wide Web um eine maschinell verarbeitbare Dimension. In ihr wird es Agenten möglich sein komplexe Aktivitäten für den menschlichen Benutzer durchzuführen und automatisch relevante Informationen im Web zu finden.

In dieser Arbeit werden wir die semantische Modellierung von Informationen im Bereich Internetfernsehen behandeln. Wir werden die gänginen Technologien und Methoden erläutern, sowie diese am Beispiel eines Prototypen für den studentischen Internetsender Zucker TV anwenden.

Abstract

The semantic web enlarges the world wide web with a machinable dimension. With this enlargement, agents will be able to perform complex tasks for human beings and to find usefull infomations in the web automatically.

In this thesis we will discuss the semantic modelling of information in a internettelevison range. We will illustrate the well-established technologies and methods as well as use them on a prototyp for the internetbroadcaster Zucker TV.

Inhaltsverzeichnis

1 Einleitung

I have a dream, diese berühmten Worte von Martin Luther King hätten auch von Tim Berners-Lee stammen können, denn als viele Leute ihn noch für seine Arbeit um das World Wide Web feierten, träumte er in Gedanken schon vom Web der nächsten Generation.

Informationen im Internet sollten nicht mehr länger nur für Menschen lesbar sein, sondern auch von Maschinen verstanden und interpretiert werden können, so dass intelligente Programme (Agenten) diese Daten verarbeiten können, um den Menschen bei komplexen Aufgaben zu unterstützen.

Wir haben uns mit dieser Arbeit vorgenommen, einen Schritt auf dem Weg hin zu dieser Vision zu gehen. An Hand eines Prototypen für einen Internetfernsehsender wollen wir zeigen in wie weit sich eine semantische Modellierung schon heute als Grundlage für eine Webanwendung nutzen lässt. Wir werden uns vor allem auf den Aspekt der Modellierung konzentrieren und zeigen, welche Vorteile semantisch modelliertes Wissen in einem Netz (Ontologie) gegenüber einer Datenhaltung in einer relationalen Datenbank bietet. Deshalb wird sich im Interface die Verwendung der semantischen Modellierung nicht unmittelbar niederschlagen, sondern sie ermöglicht uns eine einfachere und schnellere Modellierung der Wissensdomäne Film und Fernsehen. Auf die Aspekte der Wiederverwendbarkeit des modellierten Wissens,

sowie der Nutzbarkeit von externem Wissen werden, wir nur in der Theorie eingehen.

Unter diesen Vorgaben besteht unser Projekt neben einer ausführlichen Auseinandersetzung mit den theoretischen Grundlagen des *Semantic Web* aus zwei Hauptarbeitsgebieten. Zum Einen mussten wir das Wissen und die Begrifflichkeiten, die im Rahmen Film und Fernsehen genutzt werden, in Form einer Ontologie modellieren. Dieses Vokabular entstand in enger Zusammenarbeit mit Studenten und Mitarbeitern des Instituts für Theater-, Film- und Fernsehwissenschaften der Universität zu Köln. Als Experten der zubeschreibenden Domäne waren sie uns hierbei eine große Hilfe.

Das zweite Arbeitsgebiet befasst sich mit der Konzeption und Gestaltung eines Prototypen der Software, welcher die erstellte Ontologie als Navigations- und Datengrundlage nutzt. Zu Beginn mussten wir uns erst einmal in die Technologien einarbeiten, die zur Nutzung semantischer Daten benötigt werden. An Hand dieses Wissens konnte wir dann eine für uns geeignete Systemarchitektur entwerfen und parallel die Gestaltung des Benutzerinterfaces vornehmen.

Nach dieser Einführung werden wir zunächst im Kapitel *Semantic Web* die wichtigsten Technologien, Entwicklungen und Methoden im Bereich des *Semantic Web* erläutern, wobei wir auch auf dessen Probleme eingehen werden. Danach beschreiben wir den Stand der aktuellen Webanwendung von Zucker TV, wobei wir ausführlich auf die auftretenen Schwierigkeiten eingehen, um zu zeigen, wo sich Verbesserungspotential bieten. Im Anschluss daran beschreiben wir die Entwicklung eines Prototypen für Zucker TV, der sich den Vorteilen semantischer Modellierung bedient. Wir gehen vor allem auf den technischen Aufbau und die Entstehung der Ontologie ein. Schließ-

lich werden wir zum Abschluss auf unsere Arbeit zurückschauen und einen Ausblick geben, welche weiteren Entwicklungen für das Projekt möglich sind.

2 Grundlagen des Semantic Web

Tim Berners-Lee[BL01a]:

> Das *Semantic Web* ist ein Netz der Daten. Es gibt eine große Menge von Daten, die wir alle jeden Tag benutzen. Ich kann meine Bankauszüge im Web sehen, meine Photos und ich kann meine Termine in meinem Kalender sehen. Aber kann ich meine Photos in meinem Kalender sehen um zu wissen, was ich gemacht habe, während ich sie aufgenommen habe? Kann ich meine Bankauszüge in meinem Kalender sehen?
>
> Warum nicht? Weil wir kein Netz der Daten haben, weil die Daten von Anwendungen kontrolliert werden, und jede Anwendung behält ihre Daten für sich.

Die erste Erwähnung findet das *Semantic Web* in einer Publikation des *World-Wide-Web-Erfinders* Tim Berners-Lee[BL01b]. Das *Semantic Web* bezeichnet eine Erweiterung des aktuellen World Wide Web (WWW). In dieser Erweiterung sind die Information, die im Internet verfügbar sind, nicht mehr nur vom Menschen interpretierbar, sondern sie können auch von Maschinen verstanden und verarbeitet werden. Aus diesem Grund werden im *Semantic Web* nicht mehr nur Medientypen (Webseite, Film, Audiodokument, ...), sondern auch reale Objekte (Person, Ort, Organisationen ...)

beschrieben.

Ein Beispiel: Michael wird von seinem Arzt empfohlen, dass er sich einer wöchentlichen Rückengymnastik unterziehen soll. Also beauftragt er einen Agenten im Umkreis von 20 Kilometern um seinen Wohnort nach Therapiezentren zu suchen, die Termine anbieten, welche sich möglichst wenig mit den Terminen in seinem Kalender überschneiden. Für einen Menschen wären diese Informationen einfach zu finden, allerdings wäre der Zeit- und Arbeitsaufwand erheblich. Man würde zum Beispiel in den Gelben Seiten mögliche Therapiezentren ermitteln, diese einzeln anrufen und versuchen geeignete Termine zu vereinbaren.

Im *Semantic Web* würde der ganze Vorgang folgendermaßen aussehen: Der beauftragte Agent würde mittels eines Suchagentens für therapeutische Dienstleistungen in Frage kommende Zentren ermitteln. Daraufhin würde er einen Geographieagenten beauftragen, die Adressen der gefundenen Standorte mit dem Wohnort von Michael zu vergleichen und nur noch die relevanten Therapiezentren weiterzuverarbeiten. Schließlich würden die Termine aus Michaels persönlichen, sowie seinem geschäftlichen, Kalender mit den zur Verfügung stehenden Therapiezeiten verglichen, die Fahrtzeiten zu den Orten mit Hilfe eines Routenplanagenten berücksichtigt und ein individueller Therapieplan für Michael zusammen gestelllt werden. In der Vision des *Semantic Web*, könnte ein solcher Vorgang innerhalb weniger Sekunden durchgeführt werden.

Die große Neuerung, die das World Wide Web brachte, war es, voneinander unabhängige Dokumente, die jegliche Art von Information beinhalteten, miteinander zu verknüpfen. Dieser Verknüpfung wurde nicht von einer zentralen Einheit vorgenommen, sondern jeder der eine Information ins

Netz stellte, konnte diese mit beliebigen anderen Informationen verknüpfen. Durch diese Verknüpfung entstand ein Netz von Informationen, bei dem es allerdings weder Unterscheidungen bei den Inhalten der Informationen noch bei der Art der Verknüpfung (s. Abb. 2.1) gab.

Anders sieht es im *Semantic Web* (s. Abb. 2.2) aus, wo die Art der Verknüpfung, sowie der Information, durch eine definierte Bezeichnung beschrieben wird. Mit Hilfe dieser Metainformationen können dann nicht nur Menschen die Informationen interpretieren, sondern auch Programmen wird es möglich, diese Daten weiterzuverarbeiten. Das aus dem WWW bewährte System der Dezentralisierung soll auch im *Semantic Web* weiter verwendet werden, so dass es jedem möglich ist semantische Aussagen über Informationen zu treffen.

Während HTML als Sprache des WWW zur Darstellung und Präsentation der Daten dient, sollen die Sprachen des *Semantic Web* vor allem Strukturierungsinformationen und Aussagen über deren Bedeutung beinhalten.

In folgenden Abschnitt *2.1 - Architektur* werden wir genauer auf den Aufbau und die Standards des *Semantic Web* eingehen und anschließend das Gebiet *Semantic Web und Content Managment*, sowie *Probleme* behandeln. Schließlich werden wir im Abschnitt *2.6 - Ausblick* zeigen, wohin sich dieser Bereich entwickelt.

2.1 Architektur

Die Architektur des *Semantic Web* orientiert sich an dem von Tim Berners-Lee vorgestellten *Semantic Web Tower* (s. Abb. 2.3). Für einige der Ebe-

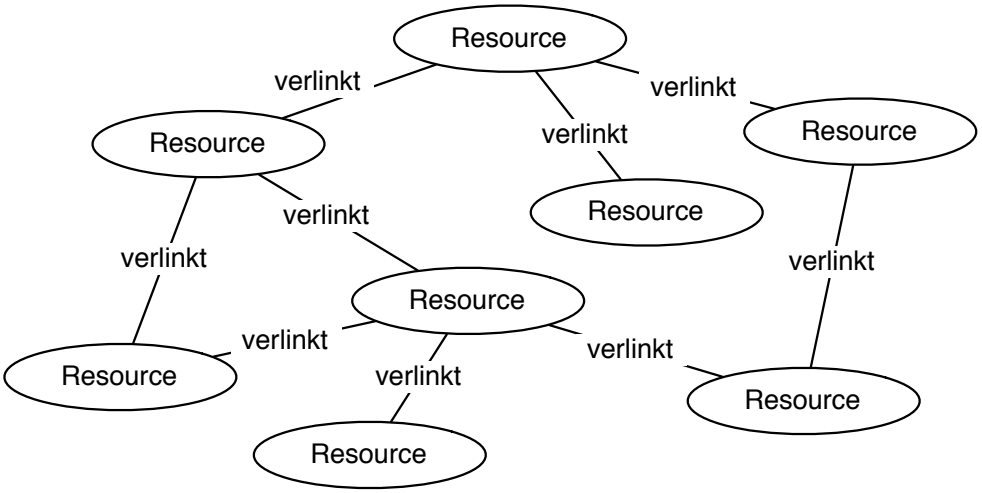

Abbildung 2.1: Linkstruktur des WWW

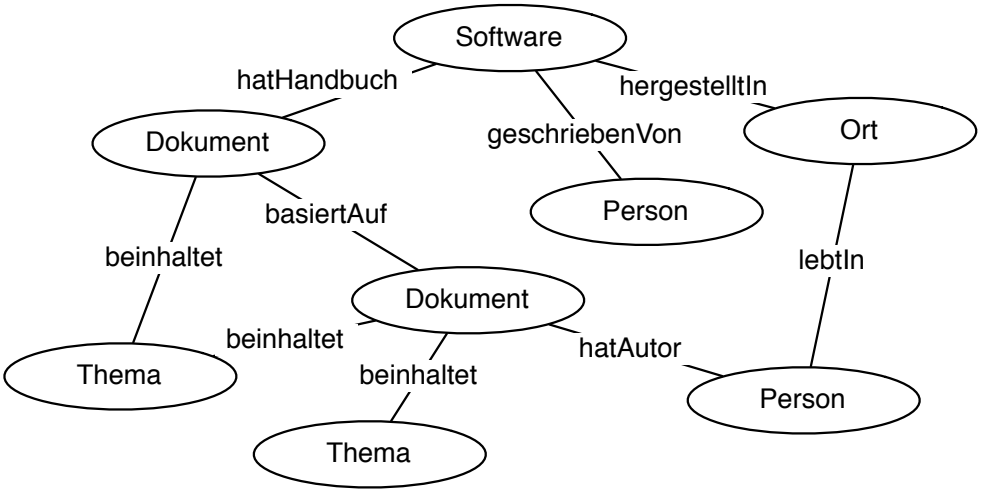

Abbildung 2.2: Linkstruktur des Semantic Web

nen gibt es bereits *Recommendations* des *World Wide Web Consortiums*[1] (W3C). In den folgenden Abschnitten werden wir etwas genauer auf die einzelnen Ebenen des *Semantic Web Tower* eingehen.

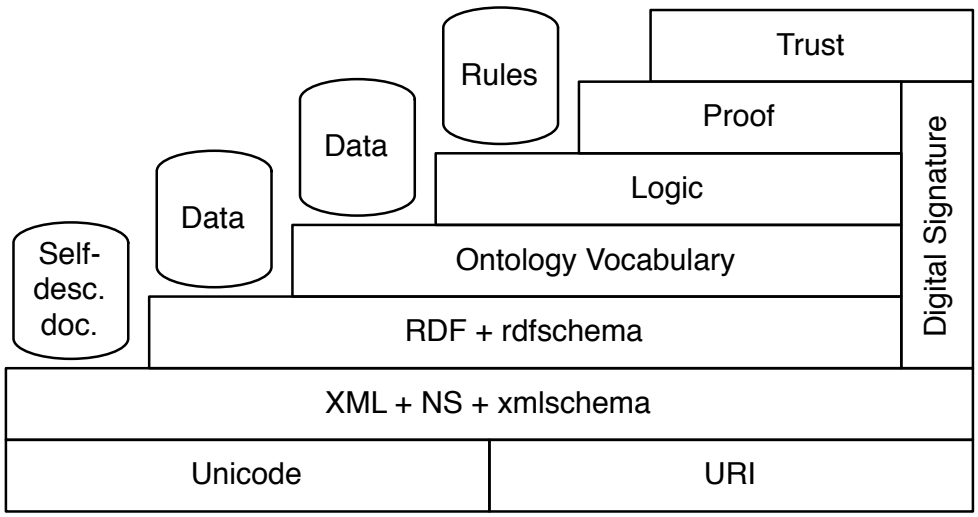

Abbildung 2.3: Der Semantic Web Tower

2.1.1 Uniform Resource Identifier

Der Uniform Resource Identifier (URI) wurde im Jahre 1994 von Tim Berners-Lee[BL94] vorgeschlagen, mit Hilfe dieses Identifikationsschemas kann jeder Ressource im Internet eine eindeutige Bezeichnung zuweisen werden. Der Begriff Ressource umfasst sowohl reale Objekte (Personen, Webseiten, ...), als auch abstrakte Konzepte (Frieden, Freiheit,...).

Da es für das *Semantic Web* wichtig ist, Konzepte eindeutig definieren und identifizieren zu können, sind URIs hierfür gute Grundlage. Meistens ähneln

[1] nichtstaatliches Gremium welches De-Facto Standards für das WWW entwickelt

sie gewöhnlichen Webadressen, zu beachten ist aber das diese *nicht* existieren müssen. Häufig finden man auf diesen Seiten Informationen über das Konzept oder das jeweilige Objekt.

2.1.2 Extensible Markup Language

Bei der Extensible Markup Language (XML) handelt es sich um eine Auszeichnungssprache für hierarchisch strukturierte Daten. Gewöhnlich wird XML in Textdateien abgelegt; es gibt aber auch einige XML Datenbanken.

Die vom W3C verabschiedete Spezifikation beschreibt XML als eine Teilmenge von SGML[2]. Mit Hilfe von Einschränkungen mittels DTD[3] oder XML Schema lassen sich wiederum anwendungsspezifische Untersprachen formulieren, wie z.B. RSS, oder SVG. Vielfach wird XML zum Austausch von Daten zwischen verschiedenen Anwendungen oder als Format für Konfigurationsdateien genutzt. Im Bereich *Semantic Web* ist die Untersprache RDF/XML zur Serialisierung der Modellierungssprachen RDF(S) und OWL definiert und standardisiert worden.

namespaces finden Verwendung wenn innerhalb eines RDF-Dokuments unterschiedliche Vokabulare genutzt werden. Denn sollten beide ein gleich benanntes aber funktionell verschiedenes Element beinhalten, können diese sonst nicht auseinander gehalten werden.

[2]Standard Generalized Markup Language
[3]Document Type Definition

2.1.3 Resource Description Framework

Resource Description Framework[MM04] (RDF) ist nicht, wie vielfach angenommen, als Format sondern mehr als eine Methode zu verstehen. Bei dieser Methode wird Wissen in definierte Strukturen aufgebrochen; dieses erreicht man, indem die Informationen als Liste von *Subjekt - Prädikat - Objekt* Aussagen definiert werden. Die Formatierung kann auf letztlich auf verschiedenste Weise erfolgen, ohne dass die Aussage hierdurch verändert wird.

In Tabelle 2.1 zeigt Beispiele für solche Aussagen; so werden die Fakten, dass Christina Karamell eine Moderatorin von Zucker TV ist und das Ens Luure eine Reportage, die aus 5 Folgen besteht und die auf Zucker TV gesendet wird, abgebildet.

Subjekt	**Prädikat**	**Objekt**
Christina Karamell	moderiert	Zucker TV
Ens Luure	läuft auf	Zucker TV
Ens Luure	ist eine	Reportage
Ens Luure	hat Folgen	5

Tabelle 2.1: Subjekt - Prädikat - Object Triple

Das Subjekt bezieht sich auf ein Gegenstand aus der wirklichen Welt, hier beispielsweise auf die Person Christina Karamell und das Format Ens Luure. Mittels des Prädikats wird die Beziehung, in der Subjekt und Objekt zueinander stehen, beschrieben. Das Objekt kann zwei unterschiedliche Ausprägungen annehmen, zum einen kann es sich wie das Subjekt, auf einen Gegenstand beziehen oder es handelt sich um einen Datenwert, zum Beispiel die die Zahl „5“.

Bei einer RDF-Aussage sollten alle Elemente der 3er-Tupel global eindeutig identifizierbar seien, denn wenn jedes Element eine eindeutige Bezeichnung erhält, so kann man klar voneinander abgrenzen, ob sich zwei Aussagen auf ein und denselben Gegenstand beziehen. Wenn wir uns wieder dem oben aufgeführten Beispiel zuwenden, sehen wir, dass es mehrere Bezüge auf das Objekt *Zucker TV* gibt, aber es nicht klar ist, ob mit beiden Objekten das Selbe gemeint ist. Dieses Problem kann durch die Verwendung von URIs (siehe Tabelle 2.2) behoben werden. Um das Beispiel übersichtlich zu halten, haben wir teils fiktionale Namensräume verwendet um die URIs abzukürzen.

Subjekt	**Prädikat**	**Objekt**
person:Christina_Karamell	tv:moderiert	http:/www.zuckertv.de
zucker:Ens_Luure	tv:läuftAufSender	http:/www.zuckertv.de
zucker:Ens_Luure	genre:isGenre	genre:Reportage
zucker:Ens_Luure	zucker:hatFolgen	“5“^^xsd:integer

Tabelle 2.2: Subjekt - Prädikat - Object Triple mit URI

Die global eindeutige Benennung der Ressourcen ermöglicht erst das Konzept der *Wiederverwendbarkeit*. Seine Stärke kann RDF erst dann ausspielen, wenn auf ein etabliertes Vokabular für bestimmte Wissensdomänen zurück gegriffen werden kann. Bevor man beispielsweise eine neue Eigenschaft formuliert, die eine Person eindeutig als Autor eines Artikels identifiziert, so sollte man überprüfen, ob es nicht vielleicht schon eine Eigenschaft gibt, welche exakt diese Beziehung modelliert. In diesem Fall würde man im Dublin Core[4] fündig werden (dc:author). Verwendet man nun diese Eigen-

[4]Dublin Core ist ein Metadaten-Schema zur Beschreibung von Dokumenten und anderen Objekten im Internet (http://de.wikipedia.org/wiki/Dublin_Core)

schaft anstatt einer eigenen, so wird die Aussage Teil des großen dezentralen Wissens, dem *Semantic Web*. Da es aber zum Einen nur wenige etablierte Vokabulare wie bspw. Dublin Core und es zum Anderen nur sehr wenige Vokabular–Verzeichnisse gibt, die man durchsuchen könnte, ist es in den meisten Fällen schwierig, eine geeignete Eigenschaft zu finden.

RDF-Schema Vor dem Hintergrund der Wiederverwendbarkeit wurde RDF-Schema (RDFS) entwickelt, um das Vokabular einer Wissensdomäne modellieren zu können. Hierfür wurden eine Reihe von Eigenschaften und das Konzept der Klassen eingeführt, bemerkenswert ist hierbei, dass RDFS, RDF Aussagen über andere RDF Aussagen sind.

Mit Hilfe von Klassen lassen sich Ressourcen verschiedenen Konzepten zuordnen, anders als in den meisten objektorientierten Umgebungen kann eine Ressource aber Mitglied beliebig vieler Klassen sein. Über die Eigenschaft rdfs:subClassOf lassen sich solche Klassenhierarchien abbilden; die Eigenschaft rdfs:type wird dagegen verwendet, um die Zugehörigkeit einer Ressource zu einer Klasse auszudrücken.

Zwei weitere wichtige RDFS-Eigenschaften sind rdfs:domain und rdfs:range. Diese beiden Eigenschaften spezifizieren den Typ der Klasse, welcher auf der Subjekt- und Objektseite einer Aussage steht. Diese Information kann in zwei Kontexten genutzt werden: zum Einen wird hier durch eine Ableitung von Wissen möglich. Da man bei Vorhandensein der rdfs:domain Eigenschaft auf die Klasse des Subjekt schließen kann, bei rdfs:range gilt dies für das Objekt der Beziehung. Zum Anderen dienen diese Eigenschaften der Dokumentation, denn sie erlauben einem fremden Entwickler besser zu verstehen, wie eine Eigenschaft zu nutzen ist.

Beachtet man nun alle vorgestellten Eigenschaften von RDF und RDFS so lässt sich daraus ein Graph wie in Abb. 2.4 erstellen. Diese (unvollständige) Abbildung der Wissensdomäne kann als eine Ontologie bezeichnet werden. Bei RDFS handelt es sich somit um eine Sprache zur Implementierung einfacher Ontologien.

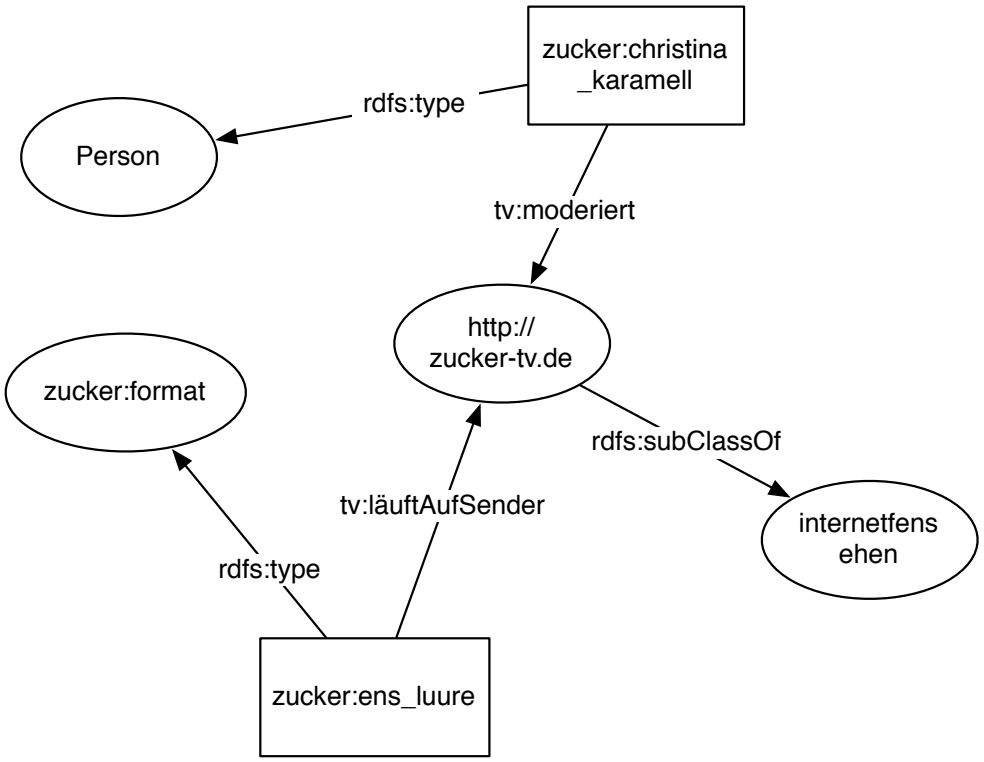

Abbildung 2.4: Zucker Beispiel Wissensdomäne

Repräsentation Wie zu Beginn erläutert, ist die Modellierung von Wissens mittels RDF(S) unabhängig von der Art der Serialisierung. Für RDF(S) haben sich drei unterschiedliche Formate herausgebildet, um das Wissen in lesbarer Form zu formatieren. Vom W3C wurde RDF/XML[Bec04] als Standard zur Abbildung von RDF-Graphen verabschiedet. Die Beispielda-

tensätze aus Tabelle 2.1 könnten in RDF/XML wie in Listing 2.1 geschrieben werden.

Listing 2.1: RDF/XML

```
<RDF>
<rdf:Description rdf:about="christina_karamell">
        <tv:moderiert>
                <http://www.zucker-tv.de>
        </tv:moderiert>
</rdf:Description>

<rdf:Description rdf:about="ens_luure">
        <tv:laeuftAuf>
                <http://www.zucker-tv.de>
        </tv:laeuftAuf>
        <zucker:genre>
                Reportage
        </zucker:genre>
</rdf:Description>
</RDF>
```

Neben dem XML basierten Standard existieren noch zwei weitere Formate, die häufiger eingesetzt werden. *Notation 3* [BL00] oder auch *N3* genannt, hebt sich vor allem dadurch hervor, dass es für den Menschen einfacher zu lesen und schreiben ist als RDF/XML. Bei *N-triples* [GB04, Ogb03] handelt es sich um ein Untersprache von N3. Der Aufbau des Formates ist einfach gehalten, pro Aussage wird jeweils eine Zeile geschrieben, welche Subjekt, Prädikat und Objekt enthält (siehe Listing 2.2).

Listing 2.2: N-triples

```
<zucker:christi...> <tv:moderiert> <http://zucker-tv.
    de> .
<zucker:christina_karamell> <rdfs:type> <zucker:person
    > .
<zucker:ens_luure> <rdfs:type> <zucker:format> .
<zucker:ens_...> <tv:luftAufSender> <http://zucke...
    > .
<http://zucke...> <rdfs:subClassOf> <tv:internetfer...
    > .
```

2.1.4 Ontologien

Dieser aus der Philosophie entliehene Begriff (Lehre des Seienden) bezeichnet einen über viele Anwendungsgebiete verteilten Forschungsgegenstand. In den frühen Neunzigern befasste sich innerhalb der Informatik nur eine kleine Forschungsgemeinschaft mit diesem Thema, sie nutzten Ontologien für die Entwicklung einer Sichtweise auf die Wissensaneignung, welche diesen Prozess als Modellierung und nicht als Transfer von Wissen beschreibt.

Gegen Ende der Neunziger erfasste auch andere Anwendungsgebiete die Erkenntnis, dass „ein konzeptuelles, aber dennoch ausführbares Modell einer Anwendungsdomäne“[SS03] für jede Art von Anwendungsszenario signifikante Vorteile zu bieten hat. Der Durchbruch von Ontologien ging einher mit der Vision des *Semantic Web* von Tim Berners Lee[BL01b]. Denn hier

sind Ontologien die Grundlage, auf der Maschinen Metadaten verstehen und weiterverarbeiten können.

Da Ontologien in den unterschiedlichsten Anwendungen und Kontexten Verwendung finden, ist eine allumfassende Definition des Begriffs nicht ohne Schwierigkeiten möglich. Unter Informatikern hat sich der Ansatz von Gruber[Gru93] „an ontology is a formal explicit specification of a shared conceptualisation“ etablieren können. Laut dieser Definition muss die Ontologie einer formalen Semantik folgen, um für Maschinen be- und verarbeitbar zu sein. Die Konzeptualisierung enthält in diesem Kontext ein gemeinsames Modell eines Aspektes der Welt, in dem Konzepte und ihre Beziehungen untereinander beschrieben werden. Da die Einigung auf ein solches Modell ein sozial geprägter Prozess ist, ergibt sich daraus die Konsequenz, dass Ontologien auf überschaubare Domänen und in manchen Fällen auch auf eine gewisse Menge von Personen beschränkt sind.

Modellierungssprachen Mit dem Aufschwung der Ontologien als Forschungsgebiet entwickelten sich eine Reihe an Sprachen um Ontologien modellieren zu können. Im Bereich der Wissenrepräsentation für Webanwendungen hat sich die vom W3C standardisierte Web Ontology Language (OWL) durchsetzen können. OWL ist keine alleinstehende Entwicklung, sondern nutzt und erweitert RDF(S) um zusätzliche Konzepte. Da Ontologiesprachen immer einen Kompromiss zwischen Entscheidbarkeit und Ausdrucksstärke finden müssen, definiert der OWL-Standard drei unterschiedliche Dialekte.

OWL Full ist zu RDF(S) syntaktisch und semantisch vollständig kompatibel. Jedoch sind Elemente enthalten, die es beispielsweise ermöglichen die Disjuktivität von Klassen auszudrücken, Klassen durch boolsche Operationen zu definieren und die Charakteristik von Eigenschaften zu beeinflussen (Inverse, Transitivität, Kardinalität). Durch die Ausdrucksstärke von OWL Full ist es allerdings nicht mehr möglich in endlicher Zeit Inferenzen einer solchen Ontologie berechnen zu lassen.

OWL DL erlaubt zwar die Nutzung aller Elemente aus OWL Full, allerdings ist ihr Gebrauch durch Regeln soweit eingeschränkt, dass die Sprache die Bedingungen der Beschreibungslogik erfüllt und somit endscheidbar wird.

OWL Lite ist wiederum eine Untermenge von OWL DL und somit voll endscheidbar, allerdings ist die Nutzung einiger Elemente eingeschränkt. So sind für Kardinalitäten nur die Werte „0“ oder „1“ erlaubt, bei boolschen Kombinationen von Klassenausdrücken darf nur die Schnittmenge mehrerer Klassen gebildet werden. Überraschenderweise ist das Ausdrücken von Disjuktivität nicht möglich. Das Ziel war es eine Sprache zu entwickeln die für die Benutzer einfacher zu verstehen, und für die Entwickler einfacher zu implementieren ist.

Da OWL als Erweiterung zu RDF(S) entwickelt wurde, besteht eine Kompatibilität zwischen den Sprachen, und es mussten bereits vorhandene Konzepte nicht erneut definiert werden. In einem OWL-Dokument wechseln sich also immer RDF(S) und OWL Elemente ab. Als Serialisierung wird auch auf die RDF-eigene Repräsentationssprache RDF/XML zurückgegriffen.

Listing 2.3: OWL/XML Auschnitt

```
<owl:Ontology rdf:about=""/>
    <rdfs:Class rdf:ID="Comedy">
        <rdfs:subClassOf rdf:resource="#Genre"/>
    </rdfs:Class>
    <rdfs:Class rdf:ID="Die_Zelle">
        <rdfs:subClassOf rdf:resource="#Format"/>
    </rdfs:Class>
    <rdfs:Class rdf:ID="Film"/>
    <owl:ObjectProperty rdf:ID="FilmIsGenre">
        <rdfs:domain rdf:resource="#Genre"/>
        <rdfs:range rdf:resource="#Film"/>
        <owl:inverseOf rdf:resource="#hasGenre"/>
 </owl:ObjectProperty>
```

2.1.5 Upper Layers

Die Ebenen im *Semantic Web Tower*, die oberhalb der Ontologien liegen, werden auch als *Upper Layer* bezeichnet. Für diese Ebenen gibt es noch keine verbindlichenen Empfehlungen des W3C, allerdings werden erste Entwicklungen in diesen Bereichen vorangetrieben.

In der *Logikschicht* definiert man Regeln zur Verarbeitung, der in der Onthologieschicht beschriebenen Inhalte. Diese werden dann in der darüberliegenden *Beweisschicht* ausgeführt und abschließend in der *Vertrauensschicht* auf ihren Vertrauenswürdigkeit hin überprüft. Dieses ist nötig, da durch die De-

zentralisierung des *Semantic Web* erst einmal jede Information den gleichen Wert besitzt und es somit keine Wahrheitsgarantie gibt.

Diese Ebenen sind die technisch größten Herausforderungen, die es auf dem Weg zu einem funktionsfähigen *Semantic Web* noch zu bewältigen gilt.

2.2 Semantic Web und Content Managment

Die im *Semantic Web* verwendeten Konzepte und Ideen bieten sich auch für kleine, geschlossene Systeme, wie sie von Content Managment Systemen (CMS) verwendet werden an. In bestehenden CMS werden im Bereich der Inhaltsverwaltung schon heute Konzepte des *Semantic Web* verwendet. So findet man in fast jedem dieser Systeme einen Schlag- und Stichwortkatalog, die man als kontrolliertes Vokabular und somit als sehr einfache Form einer Ontologie bezeichen kann.

Diestelkamp und Birkenhake[DB05] verdeutlichen dieses am Beispiel eines Musikportals. Hier könnte es einen Schlagwortkatalog zur Kennzeichnung bestimmter Artikel zu einem *Genre* zum Beispiel mit den Begriffen *Rock*, *Jazz*, *Pop* und *Hip Hop* geben. Gerade bei einer so verzweigenden Domäne wie Musikgenres wünscht man sich hier vielleicht weitergehende Möglichkeiten, möchte man beispielsweise das Genre *Grunge* hinzufügen, so hat eine Einordnung auf der Ebene der vorherigen Begriffe wenig Sinn, da *Grunge* ein Subgenre von *Rock* ist. Hier wird der Bedarf nach einer Baumstruktur für die Einordnung der Genres deutlich. Doch auch dieses Vorgehen ist noch nicht für alle Fälle ausreichend. Nimmt man ein Genre wie zum Beispiel *Crossover*, so stellt man fest, dass dieses nicht nur einen Ursprung hat. Es ist sowohl ein Subgenre von *Rock* als auch von *Hip Hop*.

Mit semantischen Techniken könnte man nicht nur diese netzartigen Strukturen modellieren, sondern auch differenzierte, bedeutungsvolle Beziehungen ziehen, so dass man die Navigations- und Suchmöglichkeiten in einem solchen System verbessern könnte.

Viele der Information, die zum Modellieren solcher sematischer Zusammenhänge benötigt werden, sind in den gängigen CMS schon implizit vorhanden. Möchte ein Autor in obigem Beispiel eine neue Plattenkritik erstellen, so wird er sich mit großer Wahrscheinlichkeit irgendeiner Art von Formular gegenüber sehen; die auszufüllenden Felder (Interpret, Titel, Plattenfirma) können als Konzepte in einer Ontologie angesehen werden. Auch einige Relationen sind schon implizit vorhanden. Allein durch ein Ausfüllen der Felder *Titel* und *Interpret* könnte eine Beziehung *hatVeröffentlicht* generiert werden. Natürlich müssen einige Relationen trotzdem von Hand angelegt werden, so dass ein Mehraufwand für die Inhaltserzeuger entsteht.

Deshalb bietet sich ein solches semantisches Content Managment System nicht für jeden Anwendungsfall an. Der anfallende Mehraufwand lohnt sich nur in Systemen, die eine große Informationsmenge und einen gewissen Informationenwert bereitstellen. In diesen Systemen könnten semantische Techniken einen großen Mehrwert für die Benutzer schaffen.

2.3 Abfragesprachen

Für semantische Datenmodelle gilt, wie auch schon für relationale, dass die modellierten Daten ohne eine passende Abfragesprache nicht verwendet werden können. Diese Tatsache führte zur Entwicklung mehrerer Sprachen, die diese Aufgabe übernehmen sollen.

SPARQL[5] gilt als Nachfolger der RDF Query Language (RDQL) und liegt seit Juni 2007 wieder als Empfehlung des W3C vor, nachdem sie diesen Status im Oktober 2006 wieder verloren hatte. Die Syntax von SPARQL erinnert sehr an die von SQL, unterscheidet sich aber in einigen Punkte hiervon.

Das erste, was in der Beispielabfrage 2.4 auffällt, ist die Anweisung Prefix, die dazu genutzt wird um, in diesem Fall, die URI http://www.zucker-tv.de mit der Zeichenkette zucker zu referenzieren. Dahinter steht das gleiche Konzept wie bei XML–Namespaces, so dass man sich das Ausschreiben der kompletten URI ersparen kann. Wäre in diesem Beispiel kein Prefix definiert, so müsste in Zeile 3 nicht zucker:isFormatMember sondern <http://www.zucker-tv.de/#isFormatMember> geschrieben stehen. Da man die Abfragesprache als Funktion verstehen kann, deren einer Eingangsparameter die Ontologie ist, erklärt sich warum auch die Abfragesprache URIs als Adressierungsschema verwendet.

Listing 2.4: Select SPARQL-Abfrage mit Where Bedingung

```
PREFIX zucker: <http://www.zucker-tv.de/#>
SELECT ?x, ?format
WHERE {?x  zucker:isFormatMember  ?format}
```

Der zweite interessante Aspekt tritt bei der Betrachtung der Where-Bedingung zu Tage. Anders als in SQL-Abfragen werden bei SPARQL keine Vergleichsoperatoren verwendet. Genau wie die tripelorientierte Modellierung des Wissens ist auch die *Where*-Bedingung nach diesem Schema aufgebaut. Die Bedingung formt eine Aussage nach dem Schema subject, predicate, object,

[5] Der Name ist rekursives Akronym und bedeutet SPARQL Protocol and RDF Query Language. http://www.w3.org/TR/rdf-sparql-query/

wobei eines oder mehrere Elemente mit Platzhaltern gefüllt werden können. Im Falle der gegebenen Abfrage sind dies subject und object das predicate ist als zucker:isFormatMember definiert. Das Auftreten des entstehenden „Musters“ wird dann innerhalb der Ontologie überprüft und führt zu einem exemplarischen Ergebnis wie in Tabelle 2.3.

x	**format**
Das Herrengedeck	Ens Luure
Gipsy Hardcore	Ens Luure
Die Quereinsteiger	Die Zelle

Tabelle 2.3: Subject, predicate, object Triple

Im Ergebnis zeigt sich die größte Schwachstelle dieser Art der Abfrage, da dem Ergebnis alle semantischen Informationen fehlen, die im Modell vorhanden gewesen sind. Durch die Veränderung der Repräsentation lässt sich auf Basis des Ergebnisses keine weitere Abfrage durchführen, wie es in vielen DBM-Systemen mittels *Sub-Selects* möglich wäre. SPARQL bietet aber auch eine Abfragevariante in der die semantischen Informationen nicht verloren gehen. Verwendet man anstatt Select das Abfragewort Construct, so wird das Ergebnis als RDF-Graph zurückgegeben.

Mit dem beschriebenen Verfahren lassen sich bestimmte Muster in der Ontologie auffinden; zusätzlich kann die Ergebnismenge noch über *Filter* weiter eingeschränkt werden. Die erste Filter-Anweisung in Listing 2.5 entfernt alle Filme mit einer Länge über 5 Minuten aus der Ergebnismenge.

Listing 2.5: Select Sparql Abfrage mit Where Bedingung

```
PREFIX zucker: <http://www.zucker-tv.de/#>
SELECT ?x
WHERE {
        ?x  zucker:laenge  ?length.
        FILTER (?length < 5)
        }
```

Neben den beschriebenen Charakteristiken werden noch weitere Möglichkeiten geboten, komplexere Abfragen zu formulieren. SPARQL ist somit ein mächtige und unverzichtbares Werkzeug beim Umgang mit sematisch modellierten Daten. Die hauptsächlich innerhalb von *Sesame* eingesetzte Sprache *Serql* bietet einen vergleichbaren Funktionsumfang, die getroffenen Aussagen gelten hierfür ebenfalls.

Der Funktionsumfang aller Abfragesprachen beschränkt sich auf das Auslesen von Daten, im Gegensatz zu SQL ist es nicht möglich Daten hinzuzufügen, bestehende zu verändern oder zu löschen. Auch hierin könnte man einen Grund sehen der die zögerliche Entwicklung von *Semantic Web*-Anwendungen erklärt.

2.4 Reasoning

Reasoning ist eine der grundlegenden Techniken des *Semantic Web*, für einige ist es sogar dessen Motor (vgl. [Bry05]). Sein Einsatzgebiet umfasst zwei Hauptaufgaben:

Erschließen nicht explizit modellierten Wissens Durch aufgestellte Ab-

leitungsregeln ist es möglich Wissen zu erschließen, welchen nicht explizit modelliert wurde. Ein einfaches Beispiel: Wir legen eine Regel fest, dass in unserer Filmwelt alle Filme, die länger als 90 Minuten sind, als *Filme mit Überlänge* gekennzeichnet werden. Wird nun ein neuer Film angelegt, bei dem die Angabe der Länge die vorgegebene Zeit überschreitet, so passiert im ersten Moment noch nichts. Erst wenn wir den Reasoner[6] auf die Ontologie anwenden, wird er auf Grund der Bedingungen eine Zuordnung unseres neuen Films zu der Klasse *Überlänge* durchführen.

Dieses Erschließen kann man jedesmal durchführen, wenn eine Änderung in der Ontologie stattfindet, und dann die Ergebnisse abspeichern, was jedoch bei Ontologien mit vielen Ableitungsregeln, die Ontologie sehr groß werden lassen kann. Die andere Möglichkeit ist es bei jeder Anfrage an die Ontologie ein Reasoning durchzuführen, was die Ontologie zwar übersichtlich, aber dafür die Abfragezeiten sehr lang werden lässt.

Überprüfung von Konsistenzregeln Durch Konsistenzregeln kann man eine Überprüfung auf Korrektheit einer Aussage ermöglichen. Dieses kann man an einem Beispiel verdeutlichen: Wenn wir in unserer Filmwelt den Begriff Trilogie verwenden, so ist dem menschlichen Benutzer klar, dass jeweils genau drei Filme zu einer Trilogie gehören müssen. Wenn wir diese Bedingung als Konsistenzregel in unsere Ontologie aufnehmen, so kann der Reasoner überprüfen, ob alle Zuordnungen korrekt sind und den Benutzer warnen, wenn dieses nicht der Fall ist.

[6] Ein Programm, welches die Ableitungs und Konsistenzregeln einer Ontologie überprüft

2.5 Probleme

So viel versprechend die Vision von Berners-Lee bezüglich des *Semantic Web* auch klingen mag. Das Konzept sieht sich einer Reihe von Problemen gegenüber, welche die Entwicklung hemmen, denn selbst sechs Jahre nach dem Artikel gibt es keine „ernstzunehmenden“ Anwendungen, die sich an den Ideen orientieren.

Da das *Semantic Web* ein Netz von dezentralem Wissen bilden soll, ergibt sich die Notwendigkeit etablierter Vokabulare, die von Entwicklern genutzt werden können. Denn solange jede Forschungs- und Anwendergruppe nur Ontologien für genau ihr Anwendungsgebiet entwickelt, bleibt die Verknüpfung von unterschiedlichen Datenbeständen aus. Zentrale Relations–Verzeichnisse könnten hier helfen, in dem sie Entwicklern eine Anlaufstelle geben, welche Beziehungen bereits modelliert wurden, um diese dann ggf. weiter zu verwenden. Aufgrund dieser Problematik haben sich in den vergangenen Jahren eine Reihe von *Mikroformaten*[Dub05] entwickelt, welche diese Lücke füllen möchten. Es handelt hierbei meist Vokabulare, die ein eng umrissenes Wissensfeld abdecken, wie z.B Adressdaten oder Termininformationen, und sich direkt, ohne Standards zu verletzen, in den (X)HTML-Quelltext einbringen lassen. Der Aufwand, der von Inhaltsproduzenten betrieben werden muss, um ihre Seite mit Informationen anzureichern, ist dementsprechend gering, was auch der Grund seien könnte, warum viele den Einsatz von Mikroformaten der „komplexen“ Modellierung in RDF(S) vorziehen.

Hinzu kommt die Problematik der momentan auf der Markt verfügbaren Werkzeuge und Frameworks, denn viele von ihnen sind noch in einem Entwicklungsstand, der eine zuverlässige Modellierung nur schwer zulässt. So ist

das von uns verwendete, an der Universität von Stanford entwickelte, Modellierungstool *Protégé*[7] eines der am weitesten entwickelten Programme auf dem Markt. Trotzdem traten mit einer erschreckenden Häufigkeit unvorhersehbare Probleme, wie neu auftauchende, niemals modellierte Klassen in der Ontologie auf. Diese Softwareprobleme könnten einer der Hauptgründe für die relativ geringe Zahl von ausgereiften Ontologien, die derzeit im Internet zu finden sind, sein.

Eine weitere Schwierigkeit ist die zur Zeit noch fehlende „Vertrauensschicht“ (siehe Abschnitt *2.1.5 - Upper Layer*). Zum Einen möchte niemand seine Daten maschinell lesbar im Netz ablegen, solange es keine Kontrolle darüber gibt, wer diese Daten zu welchen Zwecken weiterverarbeitet. Zum Anderen sollten aber auch die Agenten nicht unkontrolliert jeder Quelle vertrauen, da diese falsche oder in die Irre leitende Aussagen bereithalten könnten.

Die Zahl der Szenarios, die sich zur „unangemessenen“ Nutzung des *Semantic Web* erdenken lassen, ist groß und stellt in unseren Augen ein nicht zu unterschätzendes Risiko dar. Möglicherweise wäre der Einsatz von Signaturen ein Weg um dieser Problematik zu begegnen. Eine solche Lösung bedürfte allerdings wieder einer (oder mehrerer) zentralen Stellen, die die Verwaltung der Siganturen übernehmen, was wiederum einen Konflikt mit der offenen dezentralen Idee des *Semantic Web* hervorrufen könnte. Ein Ansatz zur Lösung dieser Schwierigkeit könnte in der Verwendung von dezentralen Identitätsmanagment- und Reputationssystemen zu finden sein. Wobei mit der Existenz produktiv nutzbarer Systeme sicher erst in einigen Jahren (wenn überhaupt) zu rechnen ist.

[7] http://protege.stanford.edu/

2.6 Ausblick

Wenn man die aktuellen Entwicklung im Bereich Webanwendungen betrachtet, sieht man zwei im Grundsatz voneinander unabhängige Entwicklungen. Auf der einen Seite haben wir das hier beschriebene *Semantic Web*, welches sich durch eine maschinell verarbeitbare Form auszeichnet, die nach strengen Regeln aufgebaut ist, und die explizite Modellierung von Wissen ermöglicht.

Auf der anderen Seite haben wir die Entwicklungen im *Web 2.0*. Unter diesem nicht exakt definierte Begriff versteht man üblicherweise eine stetig wachsende Menge von Internetangeboten, bei denen in der Regel die Beteiligung der Nutzer an der Entstehung von Inhalten im Vordergrund steht. Populäre Beispiele für solche Angebote sind Weblogs, Wikis und Medienplattformen.

Diese beiden Ansätze stehen sich auf den ersten Blick diametral gegenüber, da einmal Inhalte nach streng geregelten Vorgaben erzeugt werden, und anderseits die einfache Erzeugung von Inhalten durch Jedermann im Fokus steht. Allerdings sind sie nach Schaffert[SS06] „zwei Seiten ein und der selben Medaille“, die sich gegenseitig ergänzen können. Die Zusammenführung der Social Software (Web 2.0) und des *Semantic Web* ist für ihn die logische Weiterentwicklung der bestehenden Technologien (siehe Abb. 2.5)[8].

In diesem *Social Semantic Web* oder *Metaweb*, wie Schaffert es auch nennt gibt es zwei verschiedene Perspektiven. Auf der einen Seite sieht er die *Semantically Enabled Social Software* in der *Web 2.0 Content* mit semantischen Informationen versehen wird. Nimmt man an das Christian in einem

[8] modifiziert übernommen aus [SS06]

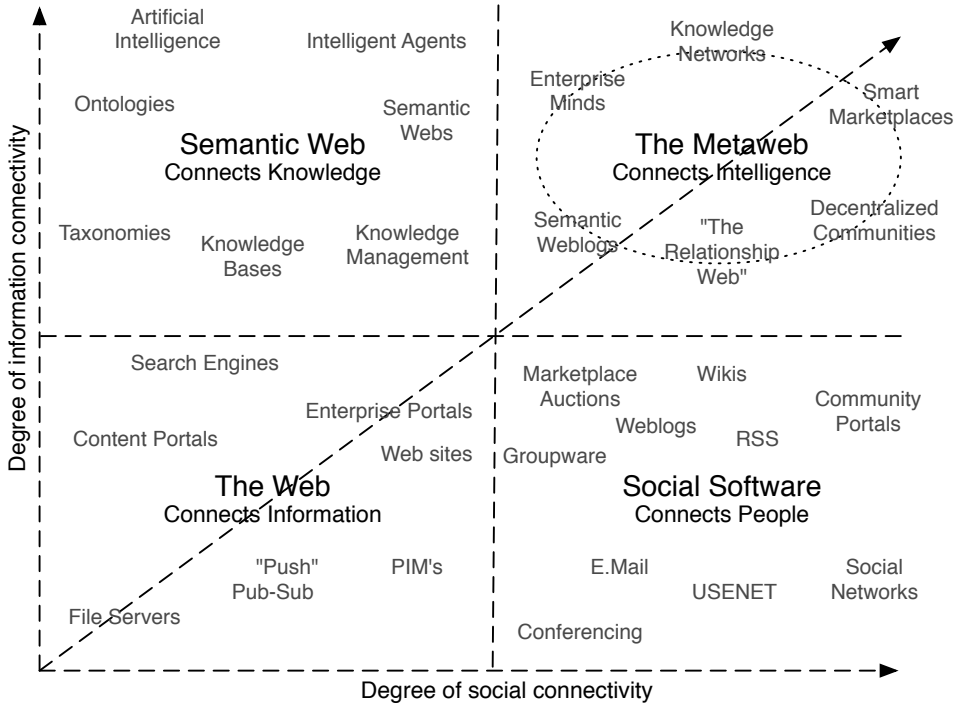

Abbildung 2.5: Das Metaweb

Artikel seinem Weblog einen Hyperlink zu Tims Weblog verwendet, so könnte man diesen mit Auszeichnung *stimmtÜberein* oder *stimmtNichtüberein* versehen. Außerdem könnten Artikel mit bestimmten Themen assoziiert werden, so dass der Benutzer nicht nur zu verwandten Themen navigieren kann, sondern explizit andere Meinungen ansteuern kann.

Andererseits könnten in kollaborativen Umgebungen Ontologien aufgebaut und gepflegt werden. Diese Perspektive nennt Schaffert *Socially Enabled Semantic Web*. Schaffert stellt einige Anwendungsgebiete für Semantic Social Software vor.

Semantic Wikis: Da jede Seite in einem Wiki mit einem bestimmten Typen assoziiert werden kann, und auch die Links zwischen den Seiten eine Bedeutung tragen, können diese gut mit semantischen Techniken beschrieben werden.

Mit diesen Metadaten können dann Informationen für bestimmte Domänen unterschiedlich präsentiert, sowie die Such- und Navigationsmöglichkeiten verbessert, werden. Außerdem ist durch diese Formalisierung ein Austausch zwischen verschiedenen Plattformen denkbar. So könnte die große Menge an Wissen, welches in Wikis abgelegt ist, für Maschinen nutzbar gemacht werden. Ein Beispiel für ein solch semantisches Wiki ist das *Semantic Media Wiki*[VKV+06]

Semantic Desktop: Bei einem Semantic Desktop geht es darum, die Idee des *Semantic Web* auf das persönliche Wissenmanagment anzuwenden. Hierbei werden kleinste inhaltliche Einheiten (z.B. ein einzelner Eintrag im Adressbuch) adressierbar gemacht und mit semantischen Annotationen versehen. So können Informationen programmübergreifend genutzt werden, und dem Benutzer kann bei komplexen Auga-

ben effiziente Unterstützung geboten werden. Die Entwicklung solcher mitdenkender Desktops wird aktuell in einigen Projekten, wie zum Beispiel *Gnowsis*[9], vorangetrieben.

[9]http://www.gnowsis.org/

3 Zucker TV - der aktuelle Stand

Zucker TV - hinter diesem Begriff verbirgt sich eine Initiative einiger TheFiFe-Studenten der Universität zu Köln. Da sich ihr Studium sehr stark auf theoretischen Aspekte konzentriert, drehten die Studenten in ihrer Freizeit kleinere Filme, kurze Serien oder Dokumentationen. Diese Werke wurden interessierten Kommilitonen dann an regelmäßigen Filmabenden vorgeführt, doch dieser Rahmen bliebt recht privat, so dass durch eine Kooperation mit dem Kölner Lokalsender *CenterTv* versucht wurde, die Filme einem größeren Publikum zu präsentieren. Da sich diese Zusammenarbeit zu Beginn als etwas schwierig herausstellte, beschloss man einen anderen Weg zu gehen und einen eigenen Internetfernsehsender zu gründen.

Nach einiger Vorbereitungs- und Konzeptionszeit wurde Zucker TV[1] im Oktober 2006 erstmals der Öffentlichkeit präsentiert. Von diesem Zeitpunkt an wurde jeden Montag ein neuer Beitrag veröffentlicht, der jeweils durch eine Moderation von Christina Karamell begleitet wurde. Mit Hilfe von Werbeaktionen, einen Ultrakurzfilmwettbewerb (5-55 Sekunden), sowie einer Kooperation mit den Videoplattform *Clipfisch*[2] und *Rheinvideo*[3] wurde um neue Zuschauer geworben, so dass Zucker TV in Spitzenzeiten über 500

[1] http://www.zuckertv.de
[2] http://www.clipfisch.de
[3] http://www.rheinvideo.de

Zuschauer pro Tag verbuchen konnte.

3.1 Technik

Da sich das Interface schnell als eine sehr interaktive Umgebung herauskristallisierte, fiel die Wahl für das Front-End auf Adobe Flash. Da laut einer aktuellen Studie[4] der Verbreitungsgrad des Player bei über 96% liegt, dürfte dessen Verwendung kein Hindernis für die Entwicklung von Zucker TV sein. Für die Datenhaltung, die neben den Filminformationen auch die Speicherung der Kommentare zu den einzelnen Filmen beinhalten sollte, entschieden wir uns für XML-Dateien, da uns eine SQL-Datenbank oder Ähnliches zu überdimensioniert erschien.

3.2 Interface

Bei der Entwicklung des Interfaces für Zucker TV trafen zwei Welten aufeinander. In der Vision der Theater-, Film- und Fernsehwissenschaftler sollte die Seite ein sehr ungewöhnliches Aussehen haben; sie stellten sich das Interface als eine Art Raum vor, in dem man an bestimmten Stellen Filme aktivieren konnte. Wir hingegen hatten uns eher an den gängigen Videoplattformen orientiert. Nach einer langen Prototyping- und Evaluationszeit wurde beschlossen, das Interface als eine Ebene von Filmstreifen anzulegen, auf denen die einzelnen Filme nach Formaten sortiert liegen sollten (s. Abb. 3.1). Um sich in diesem Raum zu bewegen, kann der Benutzer alle umliegenden Film durch anklicken in den Mittelpunkt rücken und somit eine

[4] http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

Abbildung 3.1: Die Seite des Internetfernsehsenders Zucker TV

neuen Ausschnitt des Filmraums erreichen; neben dieser Mausnavigation, kann man sich auch mittels Tastatur bewegen. Um eine weitere Navigationsform zu ermöglichen, wurden die Formate in drei Kategorien unterteilt, welche sich per Pulldownmenü ansteuern lassen.

3.3 Probleme

Nach einiger Zeit zeigte sich, dass die gewählte Interfacelösung einige Probleme mit sich brachten, was unter anderem an der Ergänzung von Funktionen lag, die in der Planung nicht angedacht waren. So ließen sich neue Navigationselemente in den statischen Aufbau des Interfaces nur schwer integrieren.

So erweist sich die im Grundsatz gute Idee, alle Filme in Streifen nebeneinander anzuordnen, mit steigender Anzahl von Filmen als zu unübersichtlich und unstrukturiert. Ein *Gelegenheitsbesucher* kann das Interface zum stöbern gut verwenden, allerdings nimmt eine bewusste Navigation mit einem bestimmten Ziel zu viel Zeit in Anspruch um effizient zu sein. Um dieses Problem einzudämmen, wurde ein Archiv mit eingebauter Suchfunktion (s. Abb 3.2) eingeführt.

Abbildung 3.2: Die Archivfunktion von Zucker TV

Ein weiteres Problem war die Einteilung der Formate in einzelne Kategorien, da sich zum Einen die Bezeichnung der Kategorien, zum Anderen auch die Zuordnung der Formate als schwierig erwiesen hat. Hinzu kamen noch Probleme, die mit der gewählten Technik zusammenhängen. Für jeden neuen Film muss die zugrunde liegende XML-Datei von Hand editiert werden, um die Einordnung des Films in den Filmraum zu gewährleisten. Auch die

Kombination Flash-PHP-XML zum Speichern der Kommentare erwies sich im Hinblick auf die Geschwindigkeit als suboptimal. Diese Geschwindigkeitsproblematik trifft auf Flash im Zusammenspiel mit größeren XML-Dateien generell zu. Auch die Gröse der zur Verfügung stehenden *Bühne* wird absehbarer Zeit nicht mehr ausreichend sein.

Aus diesen Gründen war eine Neuentwicklung des Interfaces für Zucker TV und ein Umstieg der Technik unumgänglich; in dieser neuen Version sollten die alten Fehler behoben, sowie einige neue Funktionalitäten hinzugefügt werden.

4 Umsetzung

Bei der Entwicklung des Prototypen haben wir uns für das Framework *Ruby on Rails*[1] entschieden, da dieses einen relativ einfachen Einstieg in die Materie ermöglicht. Zahlreiche automatisierte Vorgänge, das Konzept *Konvention vor Konfiguration*[2] und viele Helfermethoden erleichtern die Entwicklung und erlauben die Konzentration auf die Anwendung zu richten.

Der Hauptgrund für unsere Wahl, war allerdings die Existenz der *activeRDF*-Erweiterung, die uns den einfachen Zugriff auf unsere modellierten Daten ermöglicht. Durch die Umsetzung des MVC-Paradigmas in Rails kann die Datenbasis problemlos ausgetauscht werden, so dass man sich nicht schon zu Beginn der Entwicklung auf ein Format oder eine Modelierungsform festlegen muss. Dieses ist besonders für uns wichtig, wie man unter dem Punkt *Datenbasis* nachlesen kann.

Funktionalität

Wir haben uns bei der Umsetzung auf die Kernfunktionen der Anwendung (Navigation im Filmraum sowie die Suchfunktion) konzentriert. Die Funk-

[1] http://www.rubyonrails.org/

[2] Eine Standard Rails Anwendung kommt mit *einer* Konfigurationsdatei aus, welche die Datenbankanbindung einrichtet die in unserem Fall nicht einmal verwendet wird.

tionen, wie Kommentare, Bewertungen u.ä. werden noch nicht im Prototyp zu finden sein.

Dem Benutzer wird es auf zwei Arten ermöglicht sich durch den Filmraum zu bewegen. Neben dem aktiven Film finden sich „Pfeile“ um den jeweils älteren oder neueren Film auszuwählen. Neben dieser chronologischen Art kann der Benutzer aber auch Filme anzeigen lassen die mit dem aktuellen über dessen Eigenschaften „verwandt“ sind.

Datenbasis

Der Prototyp verwendet eine RDF(S) Version der von uns entwickelten Ontologien. Weil zur Zeit der Entwicklung kein activeRDF-Adapter zur Verfügung stand, welcher den Zugriff auf eine OWL-kompatible Datenbank erlaubt hätte. Allerdings steht ein solcher kurz vor der Fertigstellung. Sobald dieser Adapter verfügbar ist, werden wir einen Austausch der Datenbasis vornehmen können, da wir in beiden Ontologien den selben Namensraum und Eigenschaftsnamen verwendet haben.

4.1 Systemarchitektur

Die Aufgabe, die wir uns gestellt haben, war einfach und übersichtlich: „Entwickle eine Webanwendung, die ontologiebasierte Informationen verarbeitet“. Doch schon in der Vorbereitungsphase zeigte sich immer deutlicher, dass wir mit den uns bekannten Systemen und Modulen schnell an Grenzen stoßen würden.

Aus sytsemorientierter Sicht ist die Verbindung einer Anwendung mit einer

Ontologie der wichtigste Punkt in der geplanten Architektur. Es gibt eine Vielzahl von Ontologieentwicklungswerkzeugen, die aber nur dem Zweck der Entwicklung dienen, und keine Schnittstellen für externe Anwendungen bieten. Somit ist es nicht möglich, direkt auf das modellierte Wissen zuzugreifen.

Wir bewegen uns in zwei Welten, die miteinander verknüpft werden sollen, um eine funktionsfähige Anwendung zu entwickeln. Auf der einen Seite steht das objektorientierte Entwicklungsframework (in diesem Fall *Ruby on Rails*) und auf der anderen Seite die triple-orientierte Daten- und Informationsbasis in Form einer Ontologie. Eine Verknüpfung gestaltet sich somit aufwendig, da beide Seiten auf teilweise gegensätzlichen Paradigmen beruhen.

Ein Zwischenziel war es, eine Systemarchitektur zu entwickeln, die unseren Anforderungen an Nutzbarkeit, Geschwindigkeit und Erweiterbarkeit genügt. Auf diesem Weg haben wir verschiedenen Ansätze testweise implementiert oder konzeptuell durchdacht. Diese einzelnen Ideen werden im Folgenden ausführlicher behandelt.

4.1.1 Webservice

Unser erster Ansatz war eine Architektur wie sie in Abb. 4.1 dargestellt ist. Auf der Webserviceseite (Tomcat/Axis) wird die Ontologie mit Hilfe des *Jena-Frameworks* bereitgestellt, die SPARQL-Abfragen werden verarbeitet und das Ergebnis an die Rails-Anwendung zurückgegeben. Bei der Implementierung eines Testszenarios begegneten wir gleich mehreren Schwierigkeiten.

Rückgabe Da man sowohl die Größe des *ResultSet*, als auch der Aufbau einer spezifischen Abfrage im Vorhinein nicht kennt, ist es unmöglich, die für die Rückgabe benötigte XML-Struktur a priori in der WSDL-Datei zu definieren. Als Lösung ließe sich die Methode des Webservice in mehrere spezieller ausgeprägte Methoden aufbrechen, oder man gibt eine XML-Zeichenkette zurück, welche vom Client noch einmal *geparst* werden muss.

Geschwindigkeit Da für jede eintreffende Abfrage der *Reasoner* die Ontologie klassifizieren und realisieren muss, lässt sich schon bei einer kleinen Ontologie und nur einer aktiven Abfrage ein Problem mit der Antwortgeschwindigkeit des Systems feststellen.

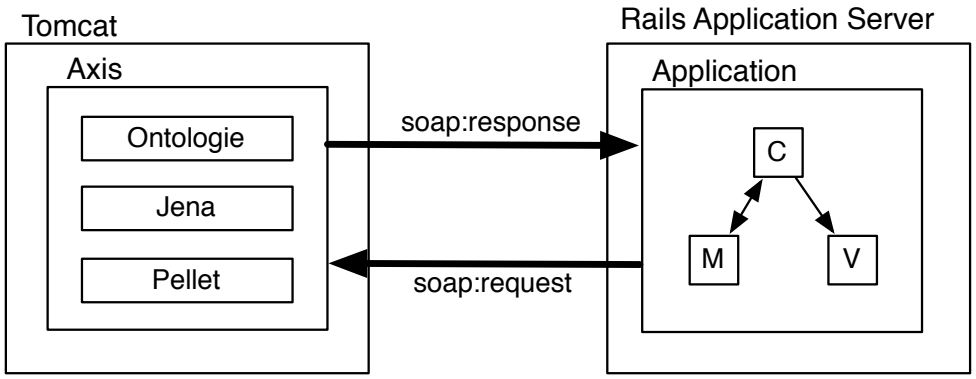

Abbildung 4.1: Webservice zwischen Webanwendung und Ontologie

Select Queries und Pellet

Möchte man innerhalb eines Programms eine SPARQL-Select Abfrage starten und für die Beantwortung eine von einem Reasoner angereicherte On-

tologie nutzen, so verwendet man einen Quellcode ähnlich dem in Listing 4.1.

Listing 4.1: SPARQL-Select Abfrage mit Jena und Pellet

```
public class PelletQueryExample {
 public static void main() throws Exception {
  String ont = "http://test.aufnahme.com/ZuckerNeu.owl
     #";

  String queryStr="SELECT ?subject ?object FROM zucker
     WHERE ?subject rdfs:subClassOf ?object";

  Query query = QueryFactory.create(queryStr);

  OntModel model = ModelFactory.createOntologyModel(
     PelletReasonerFactory.THE_SPEC );

  model.read(ont);

  QueryExecution qexec = new PelletQueryExecution(
     query, model);

  ResultSet results = qexec.execSelect();

  System.out.println(results.toString());
 }
}
```

Beim Erschaffen eines Objekts von Typ *PelletQueryExecution* wird der Pellet Reasoner in die Anfrage eingebunden. Durch den Aufruf der Methode execSelect wird die Abfrage ausgeführt. Liest man nun die Dokumention zur Pellet-API zeigt sich, dass die Klasse *PelletQueryExecution* auch eine Methode execConstruct anbietet. Modifiziert man das Programm an ein paar Stellen ergibt sich der Quellcode wie in Listing 4.2. Allerdings kommt man auf diese Weise nicht zu dem gewünschten Ergebnis, sondern nur zu einer Fehlermeldung, die mitteilt, dass eine *ClassCastException* aufgetreten ist. Jedoch konnten wir den Fehler nicht in unserem Programm auffinden und vermuten somit, dass der Fehler innerhalb der Methode execConstruct zu lokalisieren ist.

Listing 4.2: SPARQL-Construct Abfrage mit Jena und Pellet, fehlerhaft

```
public class PelletQueryExample {
 public static void main() throws Exception {
  String ont = "http://test.aufnahme.com/ZuckerNeu.owl
     #";

  String queryStr="CONSTRUCT␣{zucker:Interviews␣rdfs:
     subClassOf␣?subject␣}␣FROM␣zucker␣WHERE␣{␣?
     subject␣rdfs:subClassOf␣zucker:Interviews}";

  Query query = QueryFactory.create(queryStr);

  OntModel model = ModelFactory.createOntologyModel(
     PelletReasonerFactory.THE_SPEC );

  model.read(ont);
```

```
  QueryExecution qexec = new PelletQueryExecution(
      query, model);

  Model results = qexec.execConstruct();

  System.out.println(results.toString());
 }
}
```

Es gibt jedoch einen anderen Weg um Construct Abfragen durchzuführen ohne auf die Anbindung von Pellet verzichten zu müssen. Hierfür bedienen wir uns des Jena InfModel.

Listing 4.3: SPARQL-Construct Abfrage mit Jena InfModel und Pellet

```
public class JenaInfModel {
 public static void main(String[] args) {
  String ont = "http://test.aufnahme.com/ZuckerNeu.owl
     #";

  Reasoner reasoner = PelletReasonerFactory.
     theInstance().create();

  Model emptyModel = ModelFactory.createDefaultModel()
     ;

  InfModel model = ModelFactory.createInfModel(
     reasoner,emptyModel);
```

```
  model.read(ont);

  String queryStr = "CONSTRUCT {zucker:Interviews rdfs
      :subClassOf ?subject } FROM zucker WHERE { ?
      subject rdfs:subClassOf zucker:Interviews}";

  Query query = QueryFactory.create(queryStr);

  QueryExecution qexec = QueryExecutionFactory.create(
      query);

  Model result = qexec.execConstruct(model);

  System.out.println("----------");
  StringWriter rdfout = new StringWriter();
  result.write(rdfout);
  System.out.println(rdfout.toString());
 }
}
```

4.1.2 Ontologie-relationales Mapping

Da der erste Ansatz aufgrund zu vieler Mängel als unzureichend eingestuft wurde, haben wir einen weiteren Ansatz entwickelt. Einer der größeren Schwachpunkte der ersten Architektur war unzureichende Adaption des

Model-Paradigmas aus dem MVC-Model[3] bzw. Rails. Deshalb fokussiert sich der zweite Ansatz auf die Abbildung einer klassifizierten und realisierten Ontologie in ein relationales Datenbankschema. Die Überlegungen hierzu blieben allerdings nur rein konzeptueller Natur, da sich schnell die Ineffizienz eines solchen Systems herausstellte. Zum Einen wäre der Mapping-Prozess an sich schon eine große Herausforderung; doch zusätzlich müssten auch die Modelle (Das M in MVC) in der Rails Anwendung an das generierte Schema angepasst werden, was bereits bei ihrer Erstellung zu beachten wäre. Zudem würde man durch den Abbildung auf ein relationales Datenbankschema viel von der Aussagekraft der Ontologie verlieren, was der Idee unserer Arbeit widersprechen würde.

Die mögliche Überführung der Ontologie in eine relationale Datenbank könnte sich an einigen einfachen Regeln orientieren.

- Klassen werden zu Tabellen
- Datenwert Properties sind Spalten in der jeweiligen Klassentabelle
- subClass-Beziehnungen werden über eine zuständige Tabelle abgebildet
- Objekt Properties sind ebenfalls Zuordnungstabellen

4.1.3 activeRDF

Bei activeRDF handelt es sich um eine Erweiterung für *Ruby*, die auch unter *Ruby on Rails* genutzt werden kann. Die Idee besteht darin, den Zugriff auf RDF-Daten so einfach wie den auf eine Datenbank mittels *ac-*

[3]Das Model-View-Controller Modell, beschreibt ein System in dem Daten, Anwendunglogik und Darstellung strikt voneinander getrennt sind.

tiveRecord zu gestalten. Wie auch unter *activeRecord* werden Datensätze auf Ruby-Objekte abgebildet, damit sie dem Programmierer zur Interaktion zur Verfügung stehen. In Abb. 4.2[4] ist der Aufbau der activeRDF Architektur[ODG+05] dargestellt.

object manager bildet RDF Daten auf Stellvertreterobjekt ab. RDF(S) Klassen werden zu Ruby Klassen, RDF Ressourcen werden Ruby Objekte und RDF Eigenschaften werden zu Attributen der Ruby Objekte.

query engine wird vom *object manager* verwendet um Abfragen zur Objektmanipulation zu erzeugen. Ist unabhängig von der Datenbasis und der verwendeten Abfragesprache.

federation dient der Verwaltung der verwendeten Datenquellen. Bei der Verwendung von mehreren Datenquellen leitet er Anfragen weiter und sammelt die Ergebnisse ein.

adapters stellen der Verbindung zu einem RDF Datenspeicher her und übersetzen die Abfragen der *query engine* in für den Speicher verständliche Operationen. Es sind verschiedene Adapter das es keine einheitliche Sprache gibt um den Zugriff auf die Daten zu ermöglichen.

Zur Zeit sind Adapter für SPARQL-Endpoints, RDFLite, Redland und Sesame verfügbar. Mittels des gewählten Adapters lassen sich die gewünschten RDF-Daten über die Datei *emvironment.rb* in die Rails-Anwendung einbinden (siehe Listing 4.4).

Listing 4.4: Auszug environment.rb

```
require 'active_rdf'
```

[4]modifiziert übernommen aus [ODG+05]

```
adapter = ConnectionPool.add_data_source :type =>
 :rdflite
adapter.load "/Users/ontology/test_person_data.nt"
```

Listing 4.5 zeigt beispielhaft wie ein Rails Model gestaltet werden muss. Anders als in einem *activeRecord*-Model erbt das Model nicht von der Klasse ActiveRecord::Base, sondern in diesem Fall von der Klasse RDFS::Resource. Die Anweisung ObjectManager.construct_classes erstellt die Ruby-Klassen aus den RDF-Daten. Wie auch bei der Verwendung von activeRecord ist das Basismodel leer und enthält keine expliziten definierten Methoden. Wie bei activerocord stehen aber die dynamischen Methoden zur Abfrage von Instanzen zur Verfügung die sich nach dem Muster find_by_predicate zusammensetzen (vgl. auch Zeile 5, Listing 4.6), wobei *predicate* für den Namen einer Beziehung steht.

Listing 4.5: person.rb

```
Namespace.register :test, 'http://activerdf.org/test/'

ObjectManager.construct_classes

class TEST::Person < RDFS::Resource

end
```

Der Controller (Listing 4.6) ist das Herz der Anwendung, nachdem mit der Anweisung model : person das Model bekannt gemacht wurde, lässt es sich genau so benutzen, als ob man activeRecord verwenden würde.

Listing 4.6: say_controller.rb

```
class SayController < ApplicationController
  model :person

  def hello
    @persons = TEST::Person.find_by_eye("blue")
  end
end
```

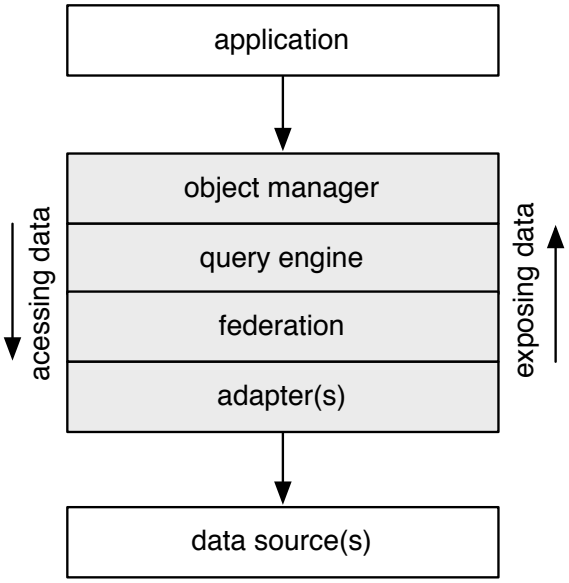

Abbildung 4.2: Schichten der activeRDF Architektur

Reasoning Support

Die activeRDF-Erweiterung an sich bietet keine Unterstützung für ein Reasoning der RDF-Daten. Dies ist allerdings von den Entwicklern so geplant, da activeRDF als Zugriffsschicht auf eine RDF-Daten konzipiert ist. activeRDF hat die Aufgabe Anfragen an eine datenhaltende Instanz zu stellen

und die Antwort in Ruby-Objekte zu transformieren. Das Reasoning sollte somit aus logischen Gründen von der RDF-Datenbank durchgeführt werden. Mit Sesame[5] steht auch ein *triple store* zur Verfügung der über Erweiterungen OWL-Reasoning unterstützt, doch der zugehörige activeRDF Adapter ist bislang noch in der Entwicklung und ließ sich im Rahmen dieser Arbeit noch nicht einsetzen.

4.2 Ontologieentwicklung

Unter der Entwicklung einer Ontologie versteht man das Erfassen einer Wissensdomäne und die anschließende Implementierung in einer Ontologiesprache. Wie dieser Prozess vom Entwickler letztendlich gestaltet wird, ist nicht durch Regeln oder Vorschriften festgelegt. Doch gibt es einige Methoden die Vorgehensweisen zur Entwicklung von Ontologien beschreiben. Diese Methoden befassen sich in erster Linie mit den formalen Aspekten der Entwicklung und greifen teils auf Methoden oder Praktiken zurück, die sich schon in der Softwareentwicklung bewährt haben.

On-To-Knowledge[SS02, SS03] wurde unter anderem an der Universität Karlsruhe entwickelt und ist in erster Linie eine Methode zur Entwicklung von Wissensmanagement Anwendungen. Da die entwickelten Anwendungen aber auf einer Ontologie aufbauen, kann eine Teilmenge dieser Methode auch zur Entwicklung von Ontologien herangezogen werden. Der erste Prozess in der Methode wird „Knowledge Meta Process“ genannt und teilt sich in fünf Schritte auf. Begonnen wird mit der *Feasibility Study* um im Vorfeld schon Probleme und Schwierigkeiten auszumachen die einen Erfolg des Sys-

[5] http://www.openrdf.org

tems behindern könnten. Diese Studie ist maßgebend für die meisten ökonomischen und technischen Entscheidungen, die zu Beginn der Entwicklung getroffen werden müssen. Nach Abschluss der ersten Phase geht die Entwicklung in die *Kickoff*-Phase über in der die eigentliche Arbeit an der Ontologie beginnt. Angelehnt an die Anforderungsanalyse bei der Softwareentwicklung wird ein *ontology requirements specification paper* (ORSD) erstellt. In diesem Dokument wird der genaue Einsatzzweck der Ontologie, Wissensquellen der Domäne, etc. festgehalten und dient dem Entwickler hauptsächlich dazu entscheiden zu können welche Konzepte in die Ontologie aufgenommen werden müssen und um die hierarchische Struktur aufzubauen. Neben dem *ORSD* wird in dieser Phase eine semi-formale Ontologie erstellt, die einer Mind-Map gleicht. Sind die Anforderungen an die Ontologie ausreichend festgelegt fließen die Ergebnisse der Phase in das *Refinement* ein. Wann der Punkt „ausreichend“ erreicht ist wird in der Regel von den Entwicklern sowie den Domänenexperten gemeinsam getroffen. Um im nächsten Schritt die semi-formale Vorlage in die *target ontology* zu überführen beginnt der Entwickler damit weitere Relationen zu der Ontologie hinzuzufügen. Dieser Vorgang ist iterativ, so wird der Entwickler erneut die Domänenexperten befragen, diese Ergebnisse in die Ontologie einarbeiten und mit dem neuen Modell wiederum die Diskussion mit den Experten der Domäne suchen. Wann von diesem Schritt der Entwicklung in den nächsten, die Evaluation, übergegangen werden kann ist häufig eine Entscheidung die vom Entwickler aufgrund seines Erfahrungsschatzes getroffen wird. Allgemein kann man jedoch sagen das man diese Phase abschließen kann wenn diese erste Ontologie genügend Essenz hat um eine prototypische Anwendung zu schreiben. Diese kann ggf. zur Evaluation genutzt werden kann, die nach On-To-Knowledge Methode in drei Typen unterschieden wird.

technology focussed evaluation Überprüfung von Syntax und Semantik der Ontologie und der verwendeten Entwicklungswerkzeuge.

user focussed evaluation Neben den technischen Aspekten steht vor allem im Vordergrund, ob die Benutzer der Anwendung ihre Aufgaben mit dem neuen System mindestens so gut wie mit dem alten bearbeiten können

fomally focussed evaluation Dieser Teil der Evaluation dient der weiteren Verfeinerung und Verbesserung der Ontologie, der OntoClean[6] Ansatz ist hier am bekanntesten und wird am häufigsten für diesen Zweck eingesetzt.

Nach der Evaluation steht in der Theorie eine anwendbare Ontologie zur Verfügung. Wie man aber Abb. 4.3[7] entnehmen kann sind die Phasen *Refinement* und *Evaluation* zyklisch angelegt d.h. in der Praxis wird die Ontologie beide Phase noch mindestens einmal komplett durchlaufen.

In der Phase *Application & Evolution* wird die Ontologie in einem Anwendungssystem eingesetzt. Die einzelnen Tätigkeiten die hierzu gehören sind im *Knowledge Process* zusammengefasst und befassen sich mit der Akquise von Wissen um dieses in die Ontologie aufzunehmen. Als *Evolution* wird der fortwährende Wartungsprozess der Ontologie bezeichnet um diese an mögliche Veränderungen anzupassen.

Wir werden uns bei der Entwicklung der Ontologie an Teilen der Methode orientieren. So werden wir die Entwicklung iterativ gestalten und eng mit den Domänenexperten zusammenarbeiten. Zudem haben wir uns dazu ent-

[6] Evaluating ontological decisions with OntoClean, *Communications of the ACM*, 45(2):61-65, 2002

[7] Abbildung entnommen aus: Handbook on Ontologies[SS03], S. 121

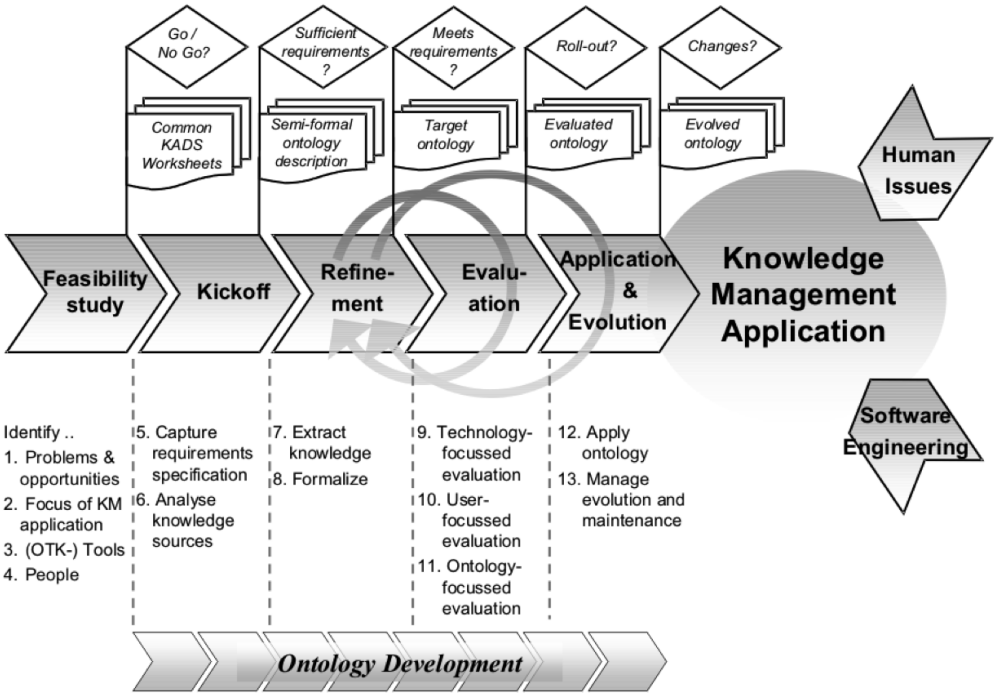

Abbildung 4.3: Knowledge Meta Process

schieden die Ontologie in zwei Teile zu zerlegen. Der erste Teil behandelt die Domäne *Filmgenre*. Der zweite Teil befasst sich mit dem spezifische Wissen bezüglich Zucker TV. Diese Trennung haben wir vorgenommen um das allgemeine Wissen, welches auch von anderen Instanzen verwendet werden könnte um Filme zu beschreiben, vom internen Wissen zu trennen. Wenn wir uns ein weiteres Filmportal vorstellen das seine Filme Genres zuordnen möchte, so könnte dieses Portal unsere Modellierung weiterverwenden. Der erste Vorteil liegt in der damit einhergehenden impliziten Verknüpfung der Datenbestände. Suchte ein Agent nun nach Filmen bspw. des Genres „Actionfilm“ unter Benutzung unserer Genreontologie würden die Filme von Zucker TV sowie des anderen Filmportals in die Ergebnismenge einfließen. Der zweite Vorteil der Wiederverwendung wäre, dass sich das imaginäre Filmportal die Arbeit sparen könnte selbst eine Modellierung dieser Domäne vorzunehmen.

4.2.1 Genreontologie

Für die Domäne Filmgenre existiert keine anerkannte Ontologie, so dass wir dieses Wissen selbst modellieren mussten, aus diesem Grund gestalteten wir die Ontologie möglichst allgemeingültig, um eine Verwendung für andere Projekte möglich zu machen. Am Anfang standen einige Gespräche mit den Studenten des Instituts für Theater- Film- und Fernsehwissenschaft, um einen Anhaltspunkt zu erarbeiten wie der Wissensbereich Filmgenre aufgebaut ist. Dabei stießen wir auf erste Schwierigkeiten. Aus filmwissenschaftlicher Sicht gibt es keine eindeutige Definition des Begriffs *Genre* und ebenso gibt es keine etablierte Menge an Genrebegriffen oder Zuordnungsregeln. Dem Großteil der Hollywoodstudios sind Genrebezeichnungen sogar

ein Dorn im Auge, da sich hierdurch existierende und kommende Filme in Raster gefasst werden, was zum Einen die Erwartungen des Zuschauers beeinflusst und den Filmen ein gewisses Maß an Individualität abspricht. Diese Ablehnungshaltung führt zu den merkwürdigen Genrekombinationen und Neukreationen so dass man in aktuellen Fernsehzeitschriften[8] mehr als **200** verschiedene Genrebezeichnungen finden kann. Obwohl der Genrebegriff bei Filmschaffenden nicht gern gesehen oder gar gemieden wird, haben wir uns entschieden das Genre als Identifikationsmerkmal in die Anwendung aufzunehmen, weil der Zuschauer anhand dieser Information dabei unterstützt wird Filme zu finden die seinem „Geschmack“ entsprechen könnten.

Auf Grundlage der Gespräche und einer Analyse der vorliegenden Informationen über die Filme, die bereits über Zucker TV gesendet wurden, haben wir einen ersten Entwurf (Abb. 4.4) der Genreontologie erstellt, die sich aber nur auf die Modellierung der Begriffshierarchie beschränkt. Dieser erste Entwurf war noch hierarchisch flach und enthielt auch noch einige Fehler, so gab es einen Knoten für „Komödie“ und „Comedy“ obwohl beide Begriffe sich auf das gleiche Genre beziehen. Ebenso wurde bei der Überarbeitung des Entwurfs eine zusätzliche Hierarchieebene „Dokumentation“ eingeführt, deren Unterelemente „Interview“ und „Reportage“ sind. Da der Entwurf noch sehr speziell durch Zucker TV geprägt war, haben wir im nächsten Schritt die Taxonomie allgemeiner gestaltet, so dass sie auch von anderen Menschen verwendet werden könnte, um Filme einem Genre zuzuordnen. Um eine möglichst allgemeingültige Modellierung zu erreichen, haben wir als „Wissensquelle“ nicht nur die TheFiFe Studenten herangezogen sondern zusätzlich ungefähr 30 weitere Menschen aus dem Bekanntenumfeld befragt und Fernsehzeitschriften analysiert. Die Masse der nun

[8] TV Spielfilm, Ausgabe 21.07 - 04.08

zur Verfügung stehenden Genrebegriffe haben wir zum Einen nach genannter Häufigkeit selektiert und dann mit Hilfe der Domänenexperten in die Ontologie eingefügt.

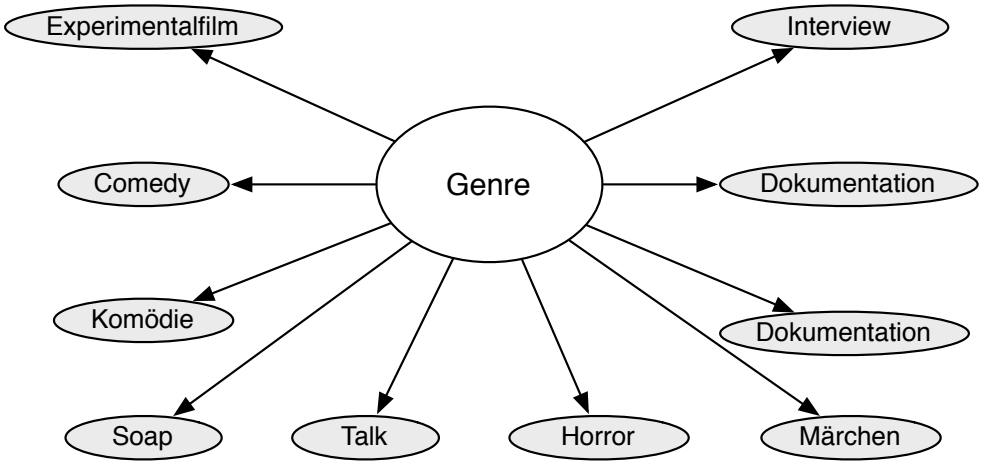

Abbildung 4.4: Erster Entwurf der Genretaxonomie

Implementierung Die nun als ausreichend gesättigt bewertete Ontologie haben wir anschließend mit Hilfe von Prot*égé* in die Sprache OWL-DL umgesetzt. Der Aufbau folgt einem einfachen Muster, unterhalb der Klasse *GenreValues* stehen mehrere Instanzen zur Verfügung welche über die Eigenschaft hatGenre einem Film zugewiesen werden können. Bei den Instanzen handelt es sich um von uns ausgemachte Kernbegriffe der Domäne Filmgenre. Anhand der Eigenschaft hatGenre wird eine Zuordnung der Filme in die Genrehierarchie vorgenommen. Existiert beispielsweise ein Film der die Attribute „Krimi“ und „Komödie“ hat so wird er dem Genre Krimi und Komödie zugeordnet. Lässt man zusätzlich die Inferenzen der Ontologie berechnen wird der Film auch dem Genre Krimikomödie zugeordnet, obwohl diese Beziehung nicht explizit modelliert wurde. Das *Reasoning* bezieht sich

nicht nur auf die Zuordnung von Filmen zu Genres, sondern auch auf die Klassenhierarchie. In der Ausgangsontologie wurde Actionkomödie nur als Unterklasse von Komödie angelegt. Ein Film ist eine Actionkomödie, wenn er mit den Attributen Action und Komödie belegt wurde. Aufgrund dieser Bedingung ist eine Actionkomödie nicht nur eine Unterklasse von Komödie, sondern auch von Actionfilm (vgl. Abb. 4.5). Durch diesen Mechanismus, der auch für die anderen Genrekombinationen greift, wird die Arbeit des Entwicklers erleichtert da, solche „Doppelzuordnungen“ nur einmal in die Ontologie einfügt werden müssen; nebenbei wird so auch eine Fehlerquelle minimiert.

Die vollständige Klassenhierarchie ist in Abb. 4.6 zu finden. Zur Veranschaulichung des vom *Reasoner* erschlossenen Wissens stellt Abb. 4.7 die Klassenhierarchie mit den berechneten Inferenzen dar.

4.2.2 Zuckerontologie

Die Ontologie der Wissensdomäne Zucker TV soll im Gegensatz zur Genreontologie das „Zucker TV spezifische Wissen“ über Filme abbilden und deren Eigenschaften beschreiben. Das Ziel ist es die gewonnene Ontologie als Datenbasis für die Internetpräsenz des Senders nutzen zu können.

Als ersten Schritt haben wir überlegt, welche Fakten einen Film beschreiben könnten. In Abb. 4.8 ist das Ergebnis dieses Brainstomings visualisiert, bei den Knoten grau unterlegten Knoten handelt es sich um die Eigenschaften, die in der aktuellen Fassung der Anwendung angezeigt und zur Suche benutzt werden können, die übrigen Knoten sind neu hinzugekommen.

Bei der Modellierung der Filmeigenschaften wollten wir dem Bereich der

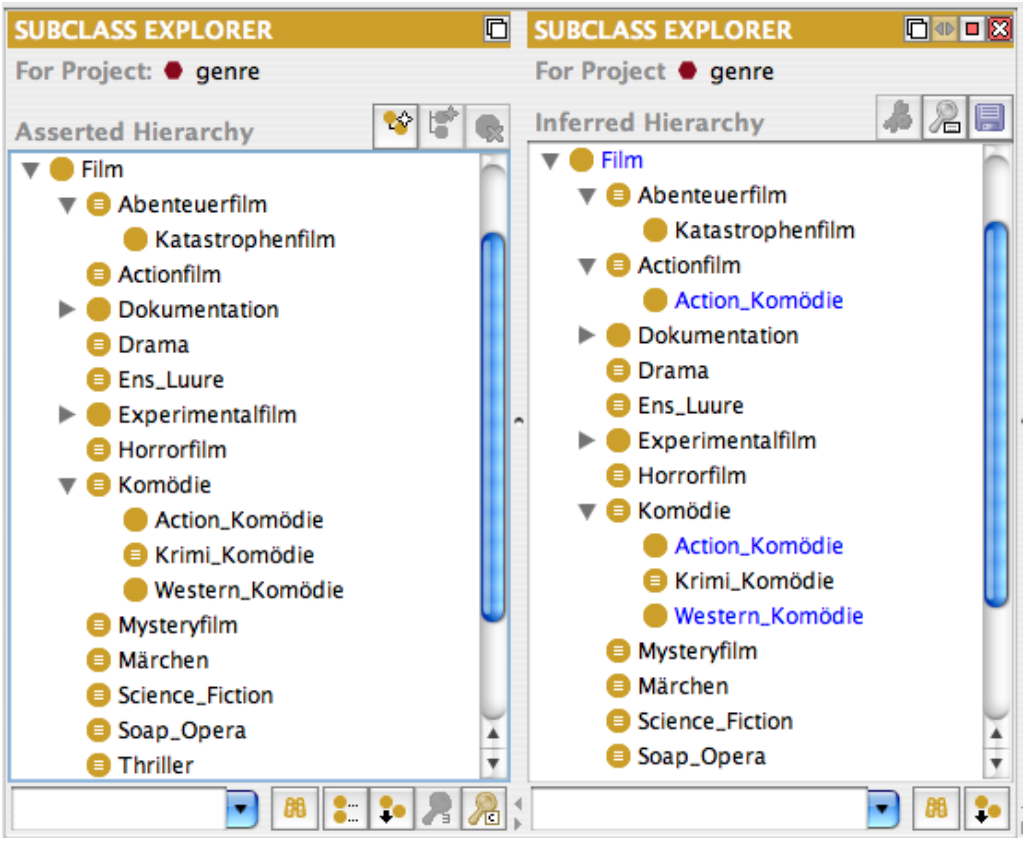

Abbildung 4.5: Polyhierarchie in der Klassifizierten Ontologie

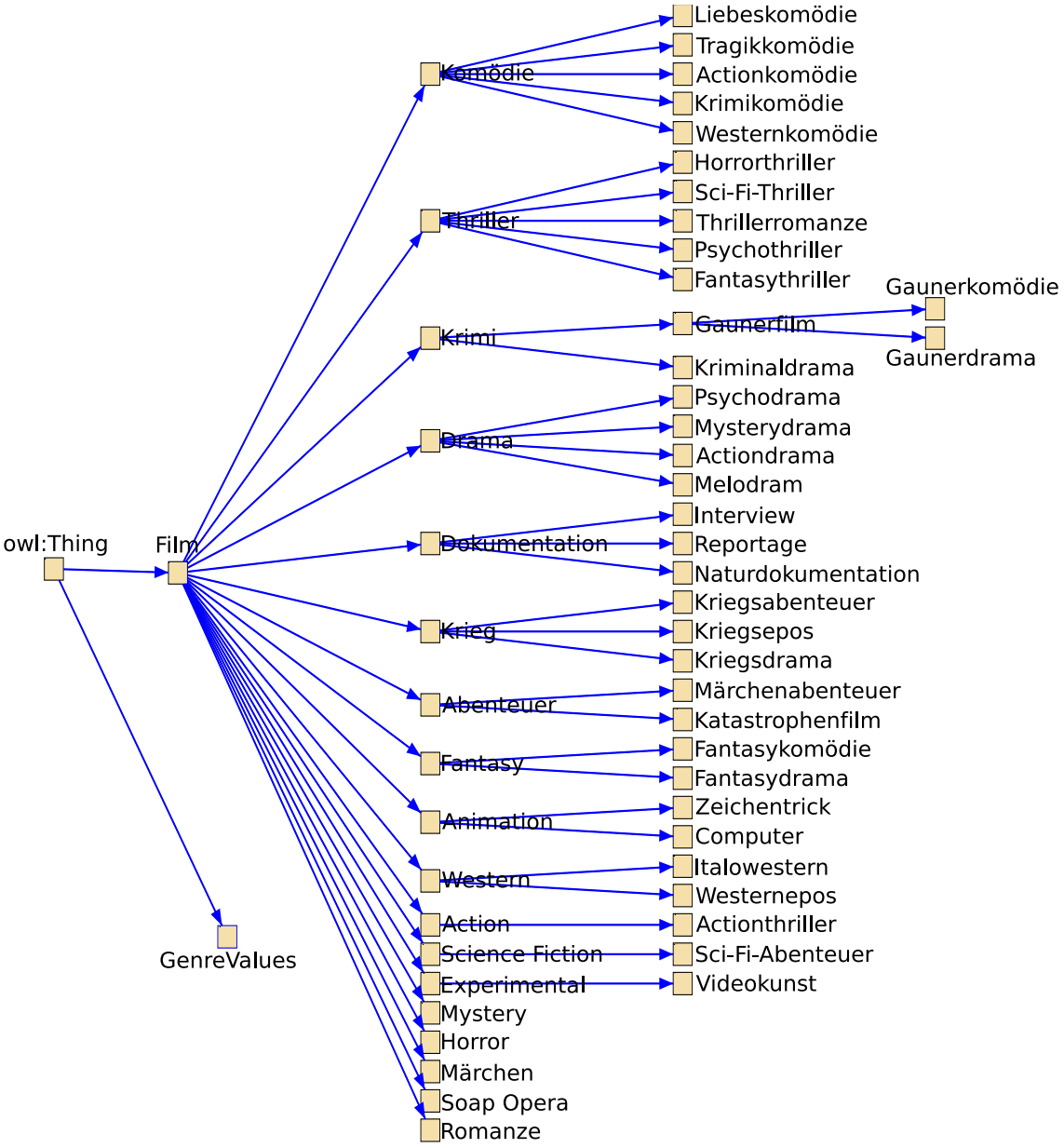

Abbildung 4.6: unklassifizierteGenreontologie

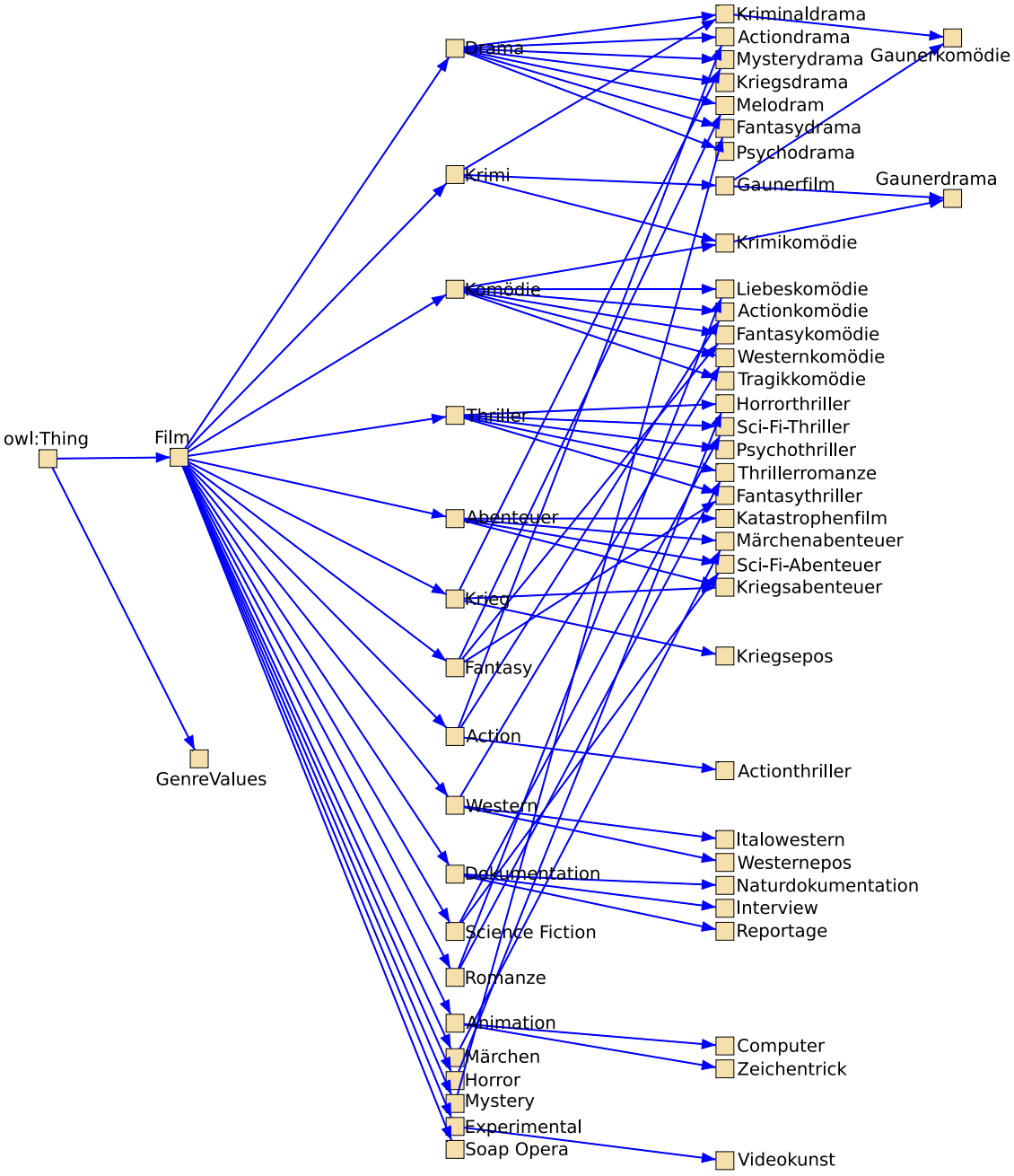

Abbildung 4.7: klassifizierte Genreontologie

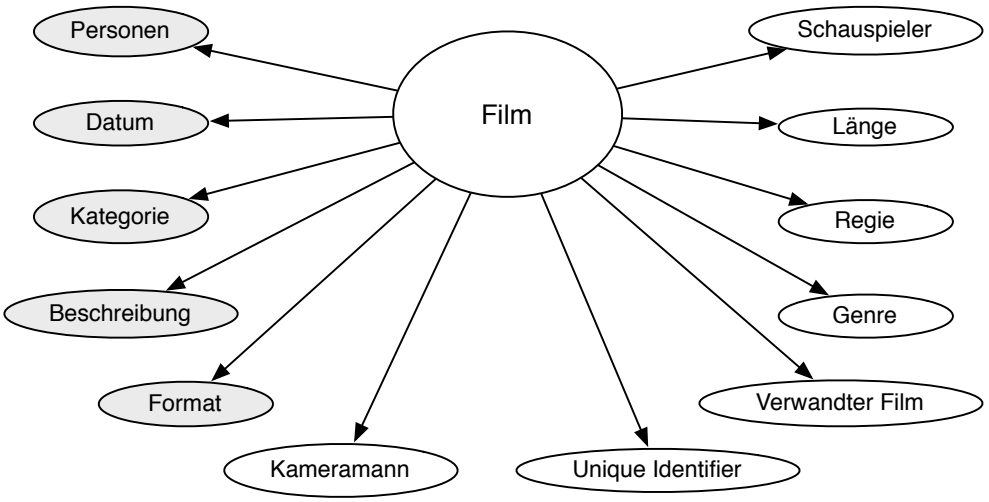

Abbildung 4.8: Filmeigenschaften

Filmschaffenden besondere Aufmerksamkeit zukommen lassen, da auf diesen Personen einer der Hauptnavigationsstränge aufbaut. Dem Benutzer soll in der späteren Anwendung die Möglichkeit gegeben werden sich weitere Filme an denen eine bestimmte Person mitgewirkt hat anzeigen zu lassen. Hierzu haben wir eine Menge an Eigenschaften eingeführt (hatRegie, hatDarsteller, hatKameramann etc.) über die Filme mit den jeweiligen Personen verbunden werden können. Die Inversen dieser Beziehungen werden zusätzlich dazu verwendet, um alle beteiligten Personen in eine Aufgabenhierarchie (vgl. Abb. 4.9) einzugliedern. Die Oberklasse aller Mitwirkenden ist „Person“, in der auch die Ursprungsinstanzen angelegt werden. Hieraus leiten sich die Klassen „Besetzung“ und „Stab“ ab, welche sich dann weiter in die einzelnen Tätigkeitsbereiche aufgliedern. Ist z.B. Person A Kameramann in Film B so ist die Person, nach erfolgtem Reasoning, nicht nur Instanz der Klasse „Person“ sondern auch der Unterklasse „Kameramann“. Um die Zuordnung vorzunehmen wird eine Einschränkung der Klassenmitglieder vorgenommen

wie sie auch in der Genreontologie verwendet wurde.

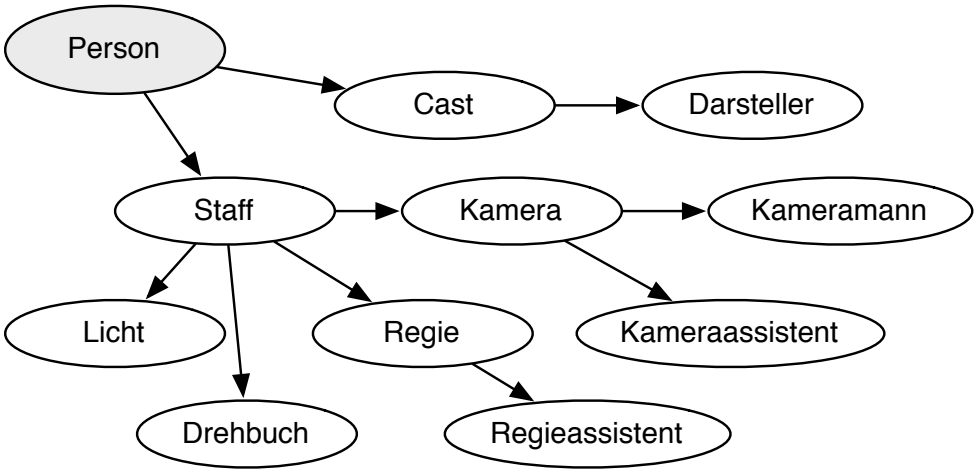

Abbildung 4.9: Ausschnitt aus der Aufgabenhierarchie

Im weiteren Gesprächen mit den TheFiFe Studenten entdeckten wir allerdings noch eine „Schwachpunkt“ der Ontologie. Von Beginn an war es der Plan die Genreontologie von der Zuckerontologie zu trennen, um eine Wiederverwendung des Genrevokabulars zu ermöglichen. Ein gleicher Schritt wäre aber auch für Beschreibung der Aufgabenbereiche bei der Filmproduktion sinnvoll gewesen, da dieses Wissen ebenso allgemein eingesetzt werden könnte um Filme zu beschreiben. Also haben wir uns dazu entschieden die Aufgabenbereiche in eine weitere Ontologie (siehe Abb. 4.10) auszulagern, welche dann wie die Genreontologie in die Zuckerontologie eingebunden wird. Somit haben wir eine stärkere Modularisierung und Wiederverwendbarkeit der Wissensbasis „Zucker TV“ erreicht, wie es in den Ideen zum *Semantic Web* angeregt wird.

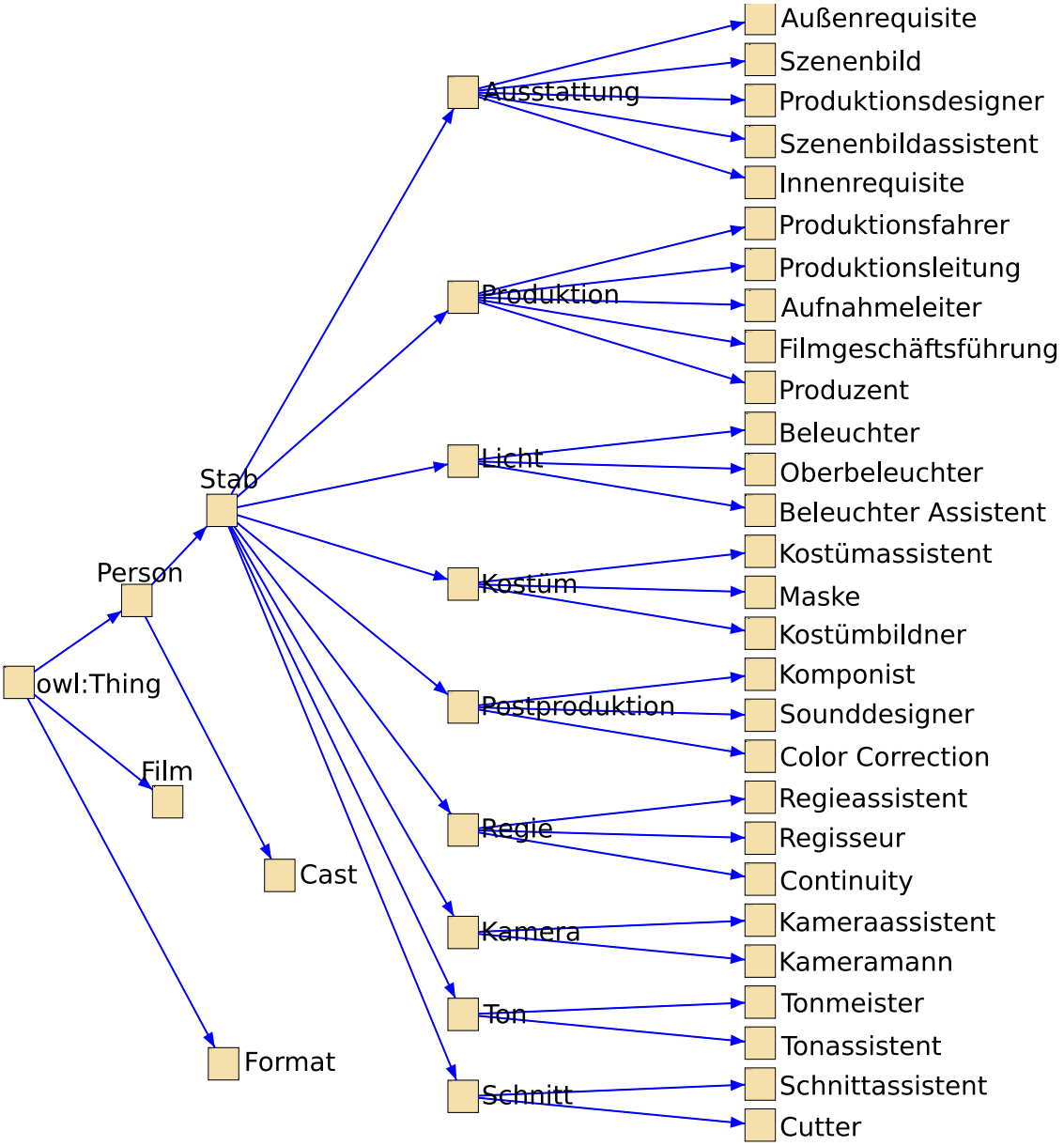

Abbildung 4.10: Tätigkeitsontologie

4.3 Interfaceentwicklung

Neben der semantischen Modellierung der Wissensdomäne, der Entwicklung einer Systemarchitektur und der Implementierung des Prototypen haben wir uns auch der Entwicklung einer neuen Benutzeroberfläche für Zucker TV gewidmet. Zwar lag dieses Thema nicht so sehr im Fokus dieser Arbeit, allerdings ermöglicht die semantische Modellierung der Datenbasis neue Möglichkeiten im Bereich der Navigation und der Suche.

4.3.1 Navigation im Filmraum

Ein Vorteil, den die semantische Modellierung bietet, besteht darin, dass Instanzen, in unserem Fall also Filme oder Personen, nicht nur auf eine Art miteinanderer verknüpft sein können, sondern das es eine Vielzahl von Möglichkeiten der Verknüpfung gibt. Dieser Mehrwert bietet sich als Navigationsgrundlage an, so dass wir uns nach einigen Iterationen von Prototypen und deren Evaluation dazu entschlossen haben, die Punkte *Explorativität* und *Kontextsensitivität* als High Level Design Goals zu definieren. Wir wollen dem Benutzer größtmögliche Freiheit bei derNavigation geben, allerdings stets im Zusammenhang mit seiner aktuellen Position im Filmraum.

Wie in Abb. 4.11 zu sehen, sind die Informationen, die zu einem Film vorhanden sind (z.B. Regisseur, Länge, usw.) gleichzeitig auch Navigationselemente. Zu jedem Punkt werden mehrere Möglichkeiten angeboten, wie man sich über eine diese Information zu anderen Filmen weiter bewegen kann. Neben statischen Informationen, werden in den nächsten Prototypen noch dynamische, vom Benutzer erstellte Informationen hinzukommen. Dieser *User Generated Content* wird sich auf „emotionale“ Einordnungen der

Filme beziehen, so können zwei Benutzer eine sehr unterschiedliche Auffassung haben, welche Filme zum Beispiel als interessant einzustufen sind. Aus diesem Grund werden solche Einordnungen auf den jeweiligen Benutzer zugeschnitten. Natürlich werden auch diese Informationen wieder als Navigationselemente zur Verfügung stehen.

Abbildung 4.11: Prototyp eines Interfaces

4.3.2 Suchen

Der zweite Hauptpunkt unserer Anwendung, in dem die semantischen Informationen eine Rolle spielen werden, ist die Suchfunktion, denn reichhaltige Informationen bieten natürlich eine gute Grundlage für eine umfangreiche Suchfunktion. Allerdings haben Tests von Experten in der Vergangenheit belegt, dass der Benutzer sehr stark von dem gängigen „Google-Suchen“ geprägt ist und die Einführung neuer Suchparadigmen nur sehr schwer möglich ist. Aus diesem Grund sahen unsere ersten Suchprototypen (siehe Abb. 4.12

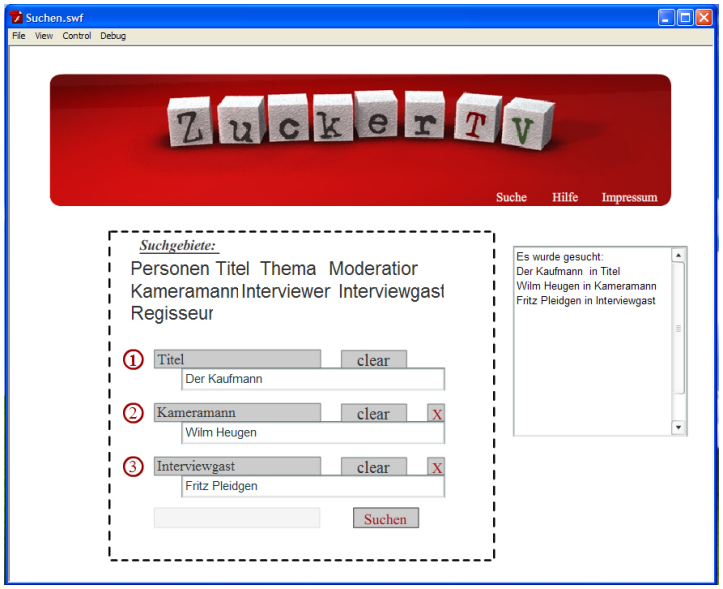

Abbildung 4.12: Prototyp der Suche

) zwar das typische Texteingabefeld vor, jedoch wollten wir dem Benutzer eine Möglichkeit geben seine Suchanfrage schon im Vorhinein genauer zu spezifizieren. Deshalb hatte er die Chance nicht nur in einem, sondern auch in mehreren Feldern Begriffe einzugeben und jeweils einen Suchbereich für jedes Eingabefeld zu definieren.

Es zeigte sich jedoch, dass die boolschen Verknüpfungen der einzelnen Felder für den Duchschnittsbenutzer nur schwer verständlich waren und dass für unseren Anwendungsfall Zucker TV, eine solche *verknüpfte* Suche unnötig komplex wäre, so dass dem Benutzer das Suchen eher erschwert als erleichtert worden wäre.

Aus diesem Grund haben wir uns dazu entschlossen, die eigentliche Suche mit nur einem Suchfeld schlicht zu gestalten. Allerdings werden wir uns bei der Präsentation der Ergebnisse wieder an dem oben erläuterten Prinzip der Explorativität orientieren, so dass der Benutzer sich aus den Sucher-

gebnissen heraus direkt weiter bewegen kann, falls der gewünschte Film nicht sofort gefunden wurde.

Außerdem werden die Suchbegriffe des Benutzers analysiert, so dass ihm weitere Hilfestellungen angeboten werden können (z.B. Suchen sie „Fritz Pleitgen“ als Filmtitel oder als Person)

5 Ausblick

Zwar ist der von uns entwickelte Prototyp noch nicht mit einem fertigen Produkt gleichzusetzen. Aber es ist ein Anfang, den es weiterzuentwickeln gilt, um möglichst bald eine Software mit semantischer Modellierung bieten zu können, die dem vollen Funktionsumfang von Zucker TV gerecht wird.

In diesem Kapitel werden wir einige interessante Entwicklungen und Projekte aus dem Bereich des *Semantic Web* ansprechen und beschreiben warum diese für Zucker TV relevant sein könnten.

5.1 Kollaborative Wissenserzeugung

Ein interessanter Ansatz, der bereits im Kapitel *2.6 - Zukunft des Semantic Web* angeschnitten wird, ist es die Erzeugung und Pflege des in Ontologien modellierten Wissens nicht mehr von einer bestimmten Person oder Instanz durchführen zu lassen, sondern diese Aufgabe zu dezentralisieren und sich des Wissens der Zuschauer zu bedienen.

Dass dieser Ansatz durchaus viel versprechend ist, zeigen Projekte wie die Wikipedia, die einzig und allein aus solchen gemeinschaftlich erzeug-

tem Wissen bestehen. Gerade im Bereich der Wikipedia gibt es momentan Bemühungen diesen gemeinschaftlichen Ansatz auch für semantische Strukturen zu verwenden und somit eine *Semantic Wikipedia* zu schaffen.

5.1.1 Semantik Wikipedia

Die Wikipedia ist die größte gemeinschaftliche erstellte Quelle für enzyklopädisches Wissen, allerdings ist dieses Wissen nur für menschliche Benutzer geeignet. Diese Schwachstelle versucht eine semantische Erweiterung des MediaWiki[1] zu beseitigen, indem das gesammelte Wissen mit sematisches Auszeichnungen versehen wird, um eine maschinelle Verarbeitung zu ermöglichen (vgl. [VKV+06]).

Der große Vorteil, den eine semantische Wikipedia bietet, besteht darin, dass man von anderen Anwendungen auf das dort vorhandene Wissen zugreifen kann (vgl. Abb. 5.1)[2]. Dieses Wissen kann dann genutzt werden, um Anwendungen mit Wissen anzureichern. So benutzt der Mediaplayer *amaroK*[3] bereits Wikipediaartikel um Informationen zu Künstlern, Alben oder Titel anzuzeigen. Sollten diese Daten mit semantischen Anreicherungen vorliegen, so könnten beispielsweise ganze Discographien von Künstler aus den in der Wikipedia verfügbaren Informationen generiert werden.

Diese automatische Weiterverwendung von Wissen würde auch für die Wikipedia an sich eine Arbeitserleichterung mit sich bringen. Denn alle Listen (bspw. aller Präsidenten der USA) werden zur Zeit noch Hand gepflegt, obwohl die jeweiligen Informationen teils schon in anderen Artikeln vorlie-

[1] http://www.mediawiki.org, die der Wikipedia zugrundeliegende Software
[2] modifiziert übernommen aus [VKV+06]
[3] http://ww.amarok.org

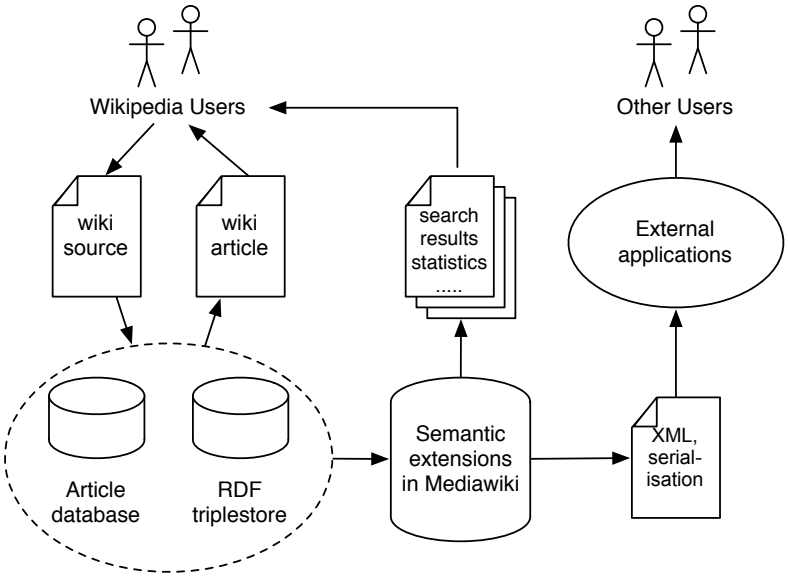

Abbildung 5.1: Architektur der Semantic MediaWiki-Erweiterung

gen. Sie können aber nicht maschinell verarbeitet werden und so ist keine Automatisierung möglich.

Ein Nachteil liegt in dem Mehraufwand, den die Benutzer der Wikipedia bei der Erstellung von Inhalten zu bewältigen haben. Doch durch ein sehr einfaches Prinzip wird dieser Aufwand klein gehalten. Zum Einen sind semantische Informationen kein Pflichtbestandteil, sondern sie können bei Bedarf benutzt werden.

Diese optionalen semantischen Annotationen lassen sich in drei Bestandteilen untergliedern. Als Erstes sind dieses Kategorien, die eine Klassifizierung eines Artikels zu seinem Inhalt vornehmen. Diese Kategorien sind bereits als Navigationsmöglichkeit in der aktuellen Wikipedia verfügbar. Der zweite Hauptbestandteil wären sogenannte *typed links*, also benannte Verknüpfungen zwischen einzelnen Artikeln. Der Mehrwert hierbei bestünde in der Wiederverwendung solcher Verknüpfungen. Man stelle sich vor, in allen Arti-

kel die sich mit Ländern beschäftigen würde der benannte Link hatHauptstadt::Stadt benutzt um eine Beziehung zwischen einem Land und seiner Hauptstadt auszudrücken. Mit Hilfe dieser Information, die der Benutzer einfach in seinen Artikel reinschreibt, könnten dann Anfragen beantwortet oder Listen von Hauptstädten erstellt werden. Der dritte Bestandteil, die *attributes*, bezeichnen einfache Eigenschaften und sind ansonsten den benannten Verknüpfungen sehr ähnlich. Möchte der Benutzer in obigem Beispiel noch die Einwohnerzahl des Landes festhalten, so schreibt er in seinen Satz einfach die Information Einwohnerzahl:=4762581. Jetzt können nicht nur Listen von Städten automatisch generiert werden, sondern sie können beispielsweise direkt nach ihrer Einwohnerzahl sortiert werden.

5.1.2 Kollaborative Wissenserzeugung für Zucker TV

Ein Problem, welches uns bei der Erstellung der Ontologien, die unser Prototyp nutzt, begegnete bestand darin, dass sich das Themengebiet Film, vor allem im Bereich Genre, nicht absolut eindeutig definieren ließ. So hat jeder Mensch eine etwas andere Vorstellung davon, welcher Film beispielsweise dem Genre Krimi zuzuordnen ist. Wird diese Zuordnung von einer Person vorgenommen, so müssen alle Benutzer mit dieser Modellierung „leben“. An dieser Stelle kommt der kollaborative Aspekt ins Spiel; wenn man die Benutzer in diffusen Wissensgebieten an den Zuordnungen beteiligen würde, wären die Ergebnisse sicherlich für eine Mehrzahl der Benutzer zufriedenstellender, als wenn dieses zentral geschehen würde.

Neben diesen Zuordnungen könnte auch die Erstellung der gesamten Ontologie von den Benutzern gemeinschaftlich erstellt werden. Da mit diesem Prinzip die Ontologie auch ständig gepflegt werden könnte, bliebe diese stets

aktuell und könnte sich an Ereignisse, die bei der Entstehung noch gar nicht bedacht werden konnten, anpassen.

5.2 Natürliche Sprache zum Suchen

Eines der Probleme, des *Semantic Web*, besteht in der Abfrage der semantischen Daten, da es keine offizelle Abfragesprache gibt, doch mittlerweile hat sich SPARQL als De-Facto-Standard (siehe Kapitel *2.3 - Abfragesprachen*) durchgesetzt. Allerdings sind zur Nutzung dieser Sprache ein Verständnis der Funktionsweise des *Semantic Web* notwendig. Die Hauptschwierigkeit besteht jedoch in dem Wissen über die Art und Weise, wie Informationen in der abgefragten Ontologie abgelegt wurden. Da diese Struktur in der Regel keinem Benutzer ausreichend bekannt ist, ist es nahezu unmöglich die gewünschten Informationen abzufragen.

Ein Ansatz um diesem Problem zu begegnen besteht in der Verwendung natürlicher Sprache, die in SPARQL-Abfragen umgewandelt wird. Der hier vorgestellte Ansatz von Lorenz Fischer[Fis06] beschreibt ein natürlichsprachliches Suchsystem für semantisch modellierte Daten.

5.2.1 NLP-Reduce

Mit dem Problem des Mapping von natürlicher Sprache auf SPARQL-Abfragen beschäftigt sich die diesem Abschnitt zugrundeliegende Arbeit. Dabei benutzt die entwickelte Software *NLP[4]-Reduce* eine abgewandelte Form der Mustererkennung in Sprache. Es wird nicht wie sonst üblich eine vorher

[4]NLP = Natural Language Processing (deutsch: Verarbeitung von natürlicher Sprache)

definiertes, statisches Muster zur Analyse der Abfragen benutzt, sondern die Muster werden dynamisch aus der zudurchsuchenden Ontologie generiert. Hierbei wird zu Beginn der Abfrageverarbeitung ein Lexikon aus den zugrundeliegenden Daten generiert (siehe Abb. 5.2)[5], dieses wird optional mit Begriffen aus der Wortdatenbank *Wordnet*[6] ergänzt. Daraufhin wird die

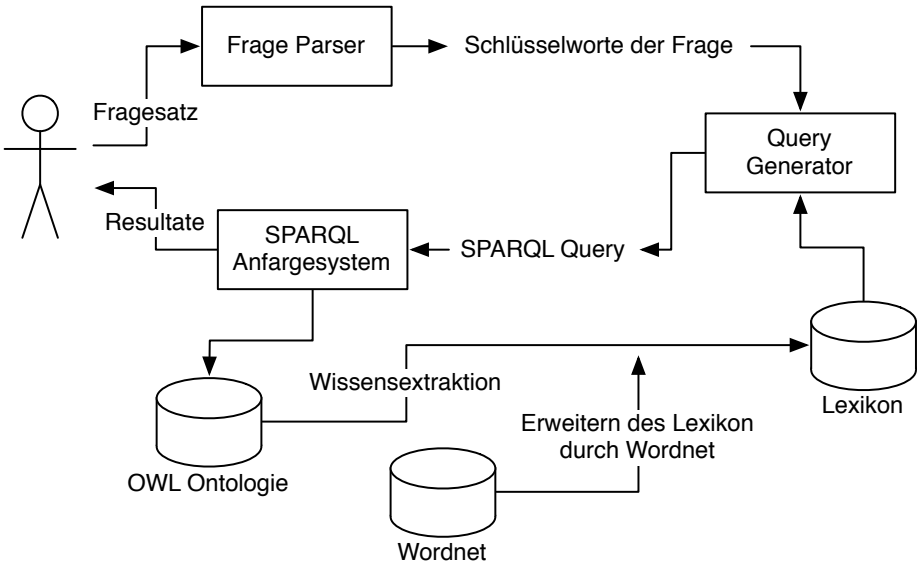

Abbildung 5.2: Systemarchitektur von NLP-Reduce

von Benutzer eingegebene Frage vorverarbeitet, bei diesem Schritt, werden unter anderem Eigennamen, die aus mehr als einem Wort bestehen (z.B. Bundesrepublik Deutschland) herausgefiltert und im Weiteren nur noch als ein Ausdruck verarbeitet. Außerdem wird an Hand der ersten Worte versucht, die Art der Frage herauszufinden; so ist bei einer Frage die mit *Wie viele* beginnt, vermutlich die Anzahl der Treffer und nicht deren Auflistung gewünscht.

[5]modifiziert übernommen aus [Fis06]
[6]http://www.wordnet.org

Der komplexeste Teil des Systems besteht aus der Abfragegenerierung. Hierbei werden die einzelnen Bestandteile der Abfrage extrahiert und mit geeigneten Eigenschaften oder Klassen der Ontologie verknüpft. Eine genau Beschreibung der Funktionsweise ist in der Arbeit von Lorenz Fischer zu finden.

5.2.2 Natürliche Sprache bei Zucker TV

Neben dem Nutzen der semantischen Modellierung für die Navigation (siehe Kapitel *4.3.1 -Navigation im Filmraum*) bietet diese auch einige Vorteile im bereich der Suche. Dieses ist eine der am häufigsten genutzten Funktionen bei Videoportalen, doch momentan wird bei diesen Suchen häufig nur der Titel des Films und eventuell noch die Filmbeschreibung durchsucht. Durch eine semantische Modellierung könnten solche Suchen deutlich effizienter durchgeführt werden, da dem Benutzer beispielsweise alternative Suchvorschläge gemacht werden könnten. Wie schon zuvor beschrieben, ist es für den Durchschnittsbenutzer nur sehr schwer möglich direkt per SPARQL Suchanfragen an die Ontologie zu stellen, so dass wir uns dazu entscheiden haben den Benutzer erst einmal nur die Parameter von vorgefertigte Abfragen(siehe Kapitel *4.3.2 - Interfaceentwicklung Suchen*) an die Ontologie verändern zulassen. Doch mit Hilfe eines Mappings von natürlicher Sprache auf SPARQL könnte man dem Benutzer ermöglichen beliebige Anfragen zu formulieren und so das Finden relevanter Informationen zu erleichtern.

6 Fazit

Zum Abschluss wollen wir auf die Arbeit zurückblicken. Als wir vor ungefähr vier Monaten begannen uns mit dem Thema *Semantic Web* auseinanderzusetzen, beschränkten sich unsere Kenntnisse auf eine grobe Vorstellung dieses Arbeitsgebietes, doch wir hielten es für ein interessantes Thema, um uns in unserer Bachelorarbeit mit dem *Semantic Web*, genauer der semantischen Modellierung zu beschäftigen.

Rückblickend kann man sagen, dass wir nicht nur eine gute Grundlage geschaffen haben, auf der wir den Prototypen weiter zu einem fertigen, einsatzfähigen Produkt entwickeln können, sondern auch einiges an neuen Fähigkeiten und Techniken gelernt haben. So haben wir nicht nur theoretische Vorgehensmodelle der Ontologieentwicklung kennen gelernt, sondern diese für drei eigenständige Ontologien auf das Einsatzgebiet und die Gegebenheiten angepasst, von denen sich sowohl die Genreontologie als auch die Filmpersonenontologie für eine allgemeine Verwendung eignen und somit eine Grundlage bilden auf der man das Wissen in dieser Domäne weiterentwickeln kann.

Außerdem ist es uns gelungen eine Systemarchitektur zu entwerfen mit der die Nutzung von semantisch modellierten Daten für eine Webanwendung einfach zu bewerkstelligen ist. Durch die Umsetzung des MVC-Paradigmas

in Ruby on Rails kann auf die semantischen Daten, wie auf Daten in einer relationalen Datenbank zugegriffen werden. In großen Projekten würde dieses bedeuten, dass sich die Entwickler der Webanwendungen selbst gar nicht mit semantischer Modellierung beschäftigen müssen, sondern einfach auf die Daten zugreifen können, so wie sie es gewohnt sind. Leider ist es mit dem momentanen Stand der Technik noch nicht möglich auf OWL-modellierte Daten zuzugreifen, doch sobald hier Adapter entwickelt worden sind, können solche Daten direkt in unserer Architektur verwendet werden und die RDF(S)-Daten ablösen.

Es hat sich außerdem gezeigt, dass die semantische Modellierung von Wissen und Informationen im Vergleich mit der Umsetzung in relationale Datenschemata noch viel Potential bietet. Denn die semantische Modellierung orientiert sich durch ihre „Sprachlichkeit“ eher an der menschlichen Denkweise. Beziehungen zwischen Gegenständen können durch Satz-ähnliche Konstrukte ausgedrückt werden anstatt diese durch *Fremdschlüssel*, *Primärschlüssel* oder „Zahlenspiele“ zu formalisieren. Begrenzende Faktoren sind mit Sicherheit die noch unzureichenden Entwicklungswerkzeuge und der erhöhte Lernaufwand. Doch wird sich diese Modellierungsform, unserer Meinung nach, mit der Zeit in einigen Anwendungsfeldern, in denen Offenheit und Flexibilität hohen Stellenwert haben, etablieren können.

Wir denken, dass wir mit dieser Arbeit eine gute Grundlage gelegt haben, auf der weitere Projekte im Bereich *Semantic Web* aufbauen können und vielleicht gelingt es in naher Zukunft jemandem mit der sich stetig weiterentwickelnden Technik weiter auf dem Weg hin zu Tim Berners-Lees Vision zu gehen Die Möglichkeiten, die das *Semantic Web* bietet sind momentan noch gar nicht abzusehen. Sehr schön formuliert hat es sein *Erfinder*

selbst[BL02]:

> Das Interessanteste am *Semantic Web* sind nicht die Dinge, die wir uns vorstellen können damit zu tun, sondern die Dinge, die wir uns noch nicht vorstellen können. Das *Semantic Web* beginnt als einfaches Diagramm mit Kreisen und Pfeilen, die Bezüge zwischen Dingen ausdrücken. Dieses Diagramm erweitert sich langsam und verknüpft sich und wird global und riesig. Das Netz, der für Menschen lesbaren Dokumente brachte eine soziale Revolution hervor. Das *Semantic Web* erreicht vielleicht das gleiche in der Welt der Computer.

Abbildungsverzeichnis

Listings

Erklärung

Ich, Lars Brillert, versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt haben, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, den 15.08.2007

Liste der von mir verfassten Kapitel:

- 2.2.2 - Extensible Markup Language
- 2.2.3 - Resource Description Framework
- 2.2.4 - Ontologien
- 2.3 - Abfragesprachen
- 2.5 - Probleme

- 4.1 - Systemarchitektur
- 4.2 - Ontologieentwicklung
- 6 - Fazit

Erklärung

Ich, Stephan Pavlovic, versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt haben, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, den 15.08.2007

Liste der von mir verfassten Kapitel:

- 1 - Einleitung
- 2 - Semantic Web Einleitung
- 2.2 - Architektur
- 2.2.1 - Uniform Resource Identifier
- 2.2.5 - Upper Layers

- 2.3 - Semantic Web und Content Managment
- 2.5 - Reasoning
- 2.6 - Ausblick
- 3 - Zucker TV
- 4.3 Interfaceentwicklung
- 5 - Ausblick
- 6 - Fazit

Literaturverzeichnis

[Bec04] Beckett, Dave: *RDF/XML Syntax Specification (Revised).* Version: 2004. http://www.w3.org/TR/rdf-syntax-grammar/, Abruf: 15.08.2007

[BL94] Berners-Lee, Tim: *RFC 1630.* Version: 1994. http://tools.ietf.org/html/rfc1630, Abruf: 15.08.2007

[BL00] Berners-Lee, Tim: *Getting into RDF & Semantic Web using N3.* Version: 2000. http://www.w3.org/2000/10/swap/Primer, Abruf: 15.08.2007

[BL01a] Berners-Lee, Tim: *Semantic Web.* Version: 2001. http://www.w3.org/2001/sw/, Abruf: 15.08.2007

[BL01b] Berners-Lee, Tim: *The Semantic Web.* Version: 2001. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&sc=I100322, Abruf: 15.08.2007

[BL02] Berners-Lee, Tim: *The Semantic Web lifts off.* Version: 2002. http://www.ercim.org/publication/Ercim_News/enw51/berners-lee.html, Abruf: 15.08.2007

[Bry05] Bry, Francois: *Reasoning ist der Motor des Semantic Web*. Version: 2005. http://www.semantic-web.at/36.51.51.article.francois-bry-reasoning-ist-der-motor-des-semantic-web.htm, Abruf: 15.08.2007

[DB05] Diestelkamp, Eike ; Birkenhake, Benjamin: *Die Semantic Web Ernuechterung*. Version: 2005. http://www.yeebase.com/fileadmin/t3n/archiv/t3n_01_semantic_web.pdf, Abruf: 15.08.2007

[Dub05] Dubinko, Micah: *What are Microformats*. Version: 2005. http://www.xml.com/pub/a/2005/03/23/deviant.html, Abruf: 15.08.2007

[Fis06] Fischer, Lorenz: *NLP-Reduce - Ein natuerlichsprachliches Suchsystem fuer Semantic Web Daten*. Version: 2006. https://www.ifi.unizh.ch/fileadmin/site/teaching/Diplomarbeiten/Abgeschlossene_Diplomarbeiten/Jahrgang_2006/Fischer_Lorenz.pdf, Abruf: 15.08.2007

[GB04] Grant, Jan ; Beckett, Dave: *RDF Test Cases*. Version: 2004. http://www.w3.org/TR/rdf-testcases/#ntriples, Abruf: 15.08.2007

[Gru93] Gruber, Thomas R.: *A Translation Approach to Portable Ontology Specifications*. Version: 1993. ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-92-71.ps.gz, Abruf: 15.08.2007

[MM04] Manola, Frank ; Miller, Eric: *RDF Primer*. Version: 2004.

http://www.w3.org/TR/rdf-primer/, Abruf: 15.08.2007

[ODG+05] Oren, Eyal et al: *ActiveRDF: Object-Oriented Semantic Web Programming.* Version: 2005. http://www.eyaloren.org/pubs/www2007.pdf, Abruf: 15.08.2007

[Ogb03] Ogbuji, Uche: *Introducing N-Triples.* Version: 2003. http://www.ibm.com/developerworks/xml/library/x-think17/index.html, Abruf: 15.08.2007

[SS02] Sure, York ; Studer, Rudi: *ON-TO Knowledge Methodology.* 2002

[SS03] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies.* Springer-Verlag, 2003. – ISBN 3–540–40834–7

[SS06] Schaffert, Steffen (Hrsg.) ; Sure, York (Hrsg.): *Semantic Social Software: Semantically Enabled Social Software or Socially Enabled Semantic Web?* Version: 2006. http://www.schaffert.eu/download/paper/Schaffert2006_SemanticSocialSoftware.pdf, Abruf: 15.08.2007

[VKV+06] Voelkel, Max et al: *Semantic Wikipedia.* Version: 2006. http://www.aifb.uni-karlsruhe.de/WBS/hha/papers/SemanticWikipedia.pdf, Abruf: 15.08.2007